LOGICAL SUBSTITUTION OF PROCESSOR CONTROL IN AN EMULATED COMPUTING ENVIRONMENT

Publication number: WO0250664
Publication date: 2002-06-27
Inventor: TRAUT ERIC P
Applicant: CONNECTIX CORP (US)
Classification:
- international: ***G06F12/10; G06F9/455; G06F9/46;*** **G06F12/10; G06F9/455; G06F9/46;** (IPC1-7): G06F9/00
- European: G06F9/455H
Application number: WO2001US49217 20011219
Priority number(s): US20000747492 20001221; US20010906392 20010716

Also published as:
WO0250664 (A3)
EP1410170 (A3)
EP1410170 (A2)
EP1410170 (A0)

Cited documents:
DE4217444
FR2587519

Report a data error here

Abstract of **WO0250664**

In an emulated computing environment, a method is provided for logically decoupling the host operating system from the processor system with respect to certain processor settings of the processor. A hypervisor of the emulation program replaces some of the processor settings of the processor with processor settings associated with software routines or data structures provided by the guest operating system; The replaced processor settings are written to memory. During this period, when the processor calls a software routine or accesses a data structure associated with the replaced processor setting, the processor will call or access a software routine or access a data structure associated with the guest operating system, bypassing the host operating system and communicating directly with the guest operating system. When the host operating system is to be recoupled to the processor, the processor settings that have been saved to memory are rewritten to the appropriate registers of the processor. During the period that the hypervisor is coupled to the processor, the page table of the hypervisor is established such that it largely mirrors the page table of the guest operating system. If a change to the page table of the guest operating system causes a conflict between the page table of the guest operating system and the page table of the hypervisor, the page table of the hypervisor is reallocated in a manner that is transparent to the guest operating system.

(19)日本国特許庁(JP)　(12)公表特許公報(A)　(11)特許出願公表番号
特表2004-531788
(P2004-531788A)
(43)公表日　平成16年10月14日(2004.10.14)

(51)Int.Cl.[7]	F I		テーマコード（参考）
G06F 12/10	G06F 12/10	553Z	5B005
G06F 9/455	G06F 12/10	505B	5B098
G06F 9/46	G06F 9/46	350	
	G06F 9/44	310A	

審査請求 未請求　予備審査請求 有　（全 51 頁）

(21)出願番号　特願2002-551693(P2002-551693)
(86)(22)出願日　平成13年12月19日(2001.12.19)
(85)翻訳文提出日　平成15年6月23日(2003.6.23)
(86)国際出願番号　PCT/US2001/049217
(87)国際公開番号　WO2002/050664
(87)国際公開日　平成14年6月27日(2002.6.27)
(31)優先権主張番号　09/747,492
(32)優先日　平成12年12月21日(2000.12.21)
(33)優先権主張国　米国(US)
(31)優先権主張番号　09/906,392
(32)優先日　平成13年7月16日(2001.7.16)
(33)優先権主張国　米国(US)

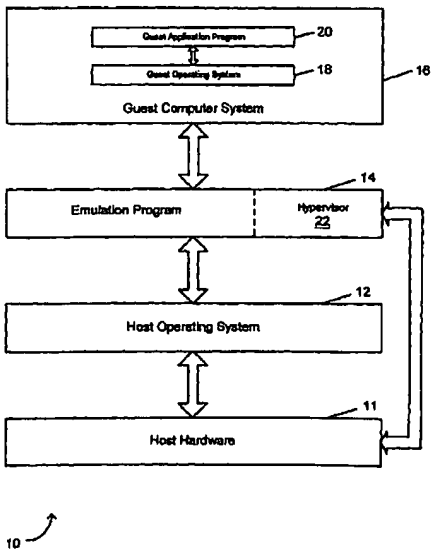
(71)出願人　391055933
コネクティクス　コーポレイション
アメリカ合衆国　９４４０３　カリフォルニア州　サン　マテオ　キャンパス　ドライブ　２９５５　スイート　１００
(74)代理人　100077481
弁理士　谷　義一
(74)代理人　100088915
弁理士　阿部　和夫
(72)発明者　エリック　ピー．トラウト
アメリカ合衆国　９４０７０　カリフォルニア州　サン　カルロス　アイリス　レーン　３

Fターム(参考)　5B005 JJ11　RR04　TT11
5B098 AA04　HH07

最終頁に続く

(54)【発明の名称】エミュレートされるコンピューティング環境におけるプロセッサ制御の論理的置換のためのシステムおよび方法

(57)【要約】
エミュレートされるコンピューティング環境において、プロセッサの一定のプロセッサ設定に関してプロセッサシステムからホストオペレーティングシステムを論理的に切り離しするための方法が提供される。エミュレーションプログラムのハイパーバイザが、プロセッサのプロセッサ設定のうちのいくつかを、ゲストオペレーティングシステムによって提供されるソフトウェアルーチンまたはデータ構造に関連するプロセッサ設定に置換する。置換されたプロセッサ設定は、メモリに書き込まれる。プロセッサがソフトウェアルーチンを呼び出すかまたは置換されたプロセッサ設定に関連するデータ構造にアクセスする時間中は、プロセッサは、ソフトウェアルーチンを呼び出すかまたはアクセスするか、あるいはゲストオペレーティングシステムに関連するデータ構造にアクセスし、ホストオペレーティングシステムをバイパスし、直接にゲストオペレーティングシステムと通信する。ホストオペレーティングシステムがプロセッサに再結合される場合は、メモリに保存されているプロセッサ設定が、プロセッサの適切なレジスタに再書き込みされる。



【特許請求の範囲】
【請求項１】
ホストコンピュータシステム内でゲストコンピュータシステムをエミュレートするための方法であって、前記ゲストコンピュータシステムはゲストオペレーティングシステムを備え、
前記ホストコンピュータシステム内でゲストオペレーティングシステムのホストとして働くためのハイパーバイザを確立するステップと、
前記ハイパーバイザ内でページテーブルを確立するステップと、
前記ゲストオペレーティングシステムの前記ページテーブルをモニタするステップと、
前記ゲストオペレーティングシステムの前記ページテーブルへの変更により、前記ハイパーバイザの前記ページテーブルと前記ゲストオペレーティングシステムの前記ページテーブルの間に競合が起きる場合は、前記ハイパーバイザの前記ページテーブルを再割り当てするステップとを含むことを特徴とする方法。
【請求項２】
ホストコンピュータシステム内でゲストコンピュータシステムをエミュレートするための方法であって、前記ゲストコンピュータシステムはゲストオペレーティングシステムを備え、前記ハイパーバイザの前記ページテーブル内に前記ハイパーバイザのコードのマッピングを含むことを除き、前記ハイパーバイザの前記ページテーブルは、一般に、前記ゲストオペレーティングシステムの前記ページテーブルをミラーリングすることを特徴とする請求項１に記載の方法。
【請求項３】
ホストコンピュータシステム内でゲストコンピュータシステムをエミュレートするための方法であって、前記ゲストコンピュータシステムはゲストオペレーティングシステムを備え、前記ゲストオペレーティングシステムの前記ページテーブルをモニタするステップは、前記ゲストオペレーティングシステムの前記ページテーブルへの書込みをモニタするステップを含むことを特徴とする請求項１に記載の方法。
【請求項４】
ホストコンピュータシステム内でゲストコンピュータシステムをエミュレートするための方法であって、前記ゲストコンピュータシステムはゲストオペレーティングシステムを備え、前記ハイパーバイザの前記ページテーブルを再割り当てするステップは、前記ハイパーバイザのコードが前記ゲストオペレーティングシステムの前記ページテーブルのマッピングされたメモリ場所とオーバラップするメモリ場所にマッピングされないように、前記ハイパーバイザの前記コードの前記マッピングを再割り当てするステップを含むことを特徴とする請求項１に記載の方法。
【請求項５】
ホストコンピュータシステム内でゲストコンピュータシステムをエミュレートするための方法であって、前記ゲストコンピュータシステムはゲストオペレーティングシステムを備え、前記ハイパーバイザの前記ページテーブル内に前記ハイパーバイザのコードのマッピングを含むことを除き、前記ハイパーバイザの前記ページテーブルは、一般に、前記ゲストオペレーティングシステムの前記ページテーブルをミラーリングし、
前記ゲストオペレーティングシステムの前記ページテーブルをモニタするステップは、前記ゲストオペレーティングシステムの前記ページテーブルへの書込みをモニタするステップを含み、
前記ハイパーバイザの前記ページテーブルを再割り当てするステップは、前記ハイパーバイザのコードが前記ゲストオペレーティングシステムの前記ページテーブルのマッピングされたメモリ場所とオーバラップするメモリ場所にマッピングされないように、前記ハイパーバイザの前記コードの前記マッピングを再割り当てするステップを含むことを特徴とする請求項１に記載の方法。
【請求項６】
エミュレートされるコンピューティング環境内で、ゲストオペレーティングシステムのペ

ージテーブルをハイパーバイザの前記ページテーブルと照合調整をするための方法であって、前記ハイパーバイザは、前記ホストコンピュータシステム上で前記ゲストオペレーティングシステムのホストとして働き、
前記ハイパーバイザの前記ページテーブルが前記ハイパーバイザのコードのためのメモリ空間の割当てを有することに加え、前記ハイパーバイザの前記ページテーブルが前記ゲストオペレーティングシステムの前記ページテーブルを大部分ミラーリングするように、前記ハイパーバイザの前記ページテーブルを確立するステップと、
前記ゲストオペレーティングシステムの前記ページテーブルへの変更のために前記ゲストオペレーティングシステムの前記ページテーブルをモニタするステップと、
前記ゲストオペレーティングシステムの前記ページテーブルへの変更により、前記ハイパーバイザの前記ページテーブルと前記ゲストオペレーティングシステムの前記ページテーブルの間に競合が起きる場合は、前記ハイパーバイザの前記ページテーブルと前記ゲストオペレーティングシステムの前記ページテーブルの間に競合がないように、前記ゲストオペレーティングシステムの前記ページテーブルを再割り当てするステップと
を含むことを特徴とする方法。

【請求項７】
エミュレートされるコンピューティング環境内で、ゲストオペレーティングシステムのページテーブルをハイパーバイザの前記ページテーブルと照合調整をするための方法であって、前記ゲストオペレーティングシステムの前記ページテーブルを再割り当てするステップは、前記ハイパーバイザのコードのページマッピングが前記ゲストオペレーティングシステム内でマッピングされたページとオーバラップしないように、前記ハイパーバイザのコードのページマッピングを再割り当てするステップを含むことを特徴とする請求項６に記載の方法。

【請求項８】
エミュレートされるコンピューティング環境内で、ゲストオペレーティングシステムのページテーブルをハイパーバイザの前記ページテーブルと照合調整をするための方法であって、前記ゲストオペレーティングシステムの前記ページテーブルをモニタするステップは、前記ゲストオペレーティングシステムの前記ページテーブルへの書込みがあった時を判断するステップを含むことを特徴とする請求項７に記載の方法。

【請求項９】
ホストコンピュータシステム内でゲストコンピュータシステムをエミュレートするための方法であって、前記ホストコンピュータシステムはプロセッサとホストオペレーティングシステムとを備え、
前記ホストシステム上でゲストオペレーティングシステムのオペレーションをエミュレートするエミュレータプログラムを実行するステップと、
前記コンピュータシステムの前記プロセッサから前記ホストオペレーティングシステムに関連するプロセッサ設定を読み取るステップと、
前記プロセッサ設定をメモリに書き込むステップと、
前記エミュレータプログラムに関連するゲストプロセッサ設定を前記プロセッサに書き込むステップと
を含み、前記ホストオペレーティングシステムは、前記エミュレータプログラムに関連する前記ゲストプロセッサ設定に関係するプロセッサ機能のために前記プロセッサから論理的に切り離しされることを特徴とする方法。

【請求項１０】
ホストコンピュータシステム内でゲストコンピュータシステムをエミュレートするための方法であって、
前記エミュレータプログラムに関連する前記プロセッサ設定は例外ベクトルであり、
前記例外ベクトルによって指し示された例外ハンドラを呼び出すステップをさらに含むことを特徴とする請求項９に記載の方法。

【請求項１１】

ホストコンピュータシステム内でゲストコンピュータシステムをエミュレートするための方法であって、
メモリから前記ホストオペレーティングシステムに関連する前記プロセッサ設定を読み取るステップと、
前記プロセッサに前記ホストオペレーティングシステムに関連する前記プロセッサ設定を書き込むステップと
をさらに含み、前記ホストオペレーティングシステムは、前記ホストオペレーティングシステムに関連する前記プロセッサ設定に関係する機能のために前記プロセッサに論理的に結合されることを特徴とする請求項９に記載の方法。

【請求項１２】
ホストコンピュータシステム内でゲストコンピュータシステムをエミュレートするための方法であって、
前記エミュレータプログラムによって要求された機能を実施するために前記ホストオペレーティングシステムを前記プロセッサに一時的に論理的に再結合するステップと、
前記ホストオペレーティングシステムが前記エミュレータプログラムによって要求された機能を完了した後、前記エミュレータプログラムに関連する前記プロセッサ設定に関係する機能のために前記エミュレータプログラムを前記プロセッサに論理的に再結合するステップと
をさらに含むことを特徴とする請求項９に記載の方法。

【請求項１３】
ホストコンピュータシステム内でゲストコンピュータシステムをエミュレートするための方法であって、前記エミュレータプログラムは、前記ホストオペレーティングシステム上でアプリケーションプログラムとして動作することを特徴とする請求項９に記載の方法。

【請求項１４】
ホストコンピュータシステム内でゲストコンピュータシステムをエミュレートするための方法であって、
前記エミュレータプログラムに関連する前記プロセッサ設定は割込ルーチンポインタであり、
前記割込ルーチンポインタによって指し示された割込ルーチンを呼び出すステップをさらに含むことを特徴とする請求項９に記載の方法。

【請求項１５】
ホストコンピュータシステム内でゲストコンピュータシステムをエミュレートするための方法であって、
前記エミュレータプログラムに関連する前記プロセッサ設定は例外ベクトルテーブルに対するポインタであり、
前記エミュレータプログラムに関連する前記プロセッサ設定を使用して前記例外ベクトルテーブルにアクセスするステップをさらに含むことを特徴とする請求項９に記載の方法。

【請求項１６】
ホストコンピュータシステム内でゲストコンピュータシステムをエミュレートするための方法であって、
前記エミュレータプログラムに関連する前記プロセッサ設定は前記ゲストコンピュータシステムに関連するページテーブルに対するポインタであり、
前記エミュレータプログラムに関連する前記プロセッサ設定を使用して前記ゲストコンピュータシステムの前記ページテーブルにアクセスするステップをさらに含むことを特徴とする請求項９に記載の方法。

【請求項１７】
コンピュータシステムのプロセッサから前記コンピュータシステムのホストオペレーティングシステムを論理的に切り離しするための方法であって、
前記ホストコンピュータシステム上のアプリケーションプログラムとして、ゲストオペレーティングシステムのオペレーションをエミュレートするエミュレータプログラムを実行

するステップを含み、前記ゲストオペレーティングシステムは、前記コンピュータシステムの前記プロセッサのスーパバイザ状態に関連する１組の機能を有し、
さらに、前記ホストオペレーティングシステムが、交換されたプロセッサ設定に関連するプロセッサオペレーションに関して前記プロセッサから論理的に切り離しするように、前記プロセッサ内で第１の交換ステップの一部として、前記ゲストオペレーティングシステムの機能に関連するプロセッサ設定と前記プロセッサのスーパバイザ状態とを交換するステップと、
前記ホストオペレーティングシステムが、前記交換されたプロセッサ設定に関連するプロセッサオペレーションに関して前記ホストオペレーティングシステムに論理的に再結合するように、前記プロセッサ内で第２の交換ステップの一部として、前記ホストオペレーティングシステムの機能に関連するプロセッサ設定を交換するステップと
を含むことを特徴とする方法。

【請求項１８】
コンピュータシステムのプロセッサから前記コンピュータシステムのホストオペレーティングシステムを論理的に切り離しするための方法であって、前記ゲストオペレーティングシステムの機能に関連するプロセッサ設定は、前記ゲストオペレーティングシステムによって提供される例外ハンドラに対するポインタであることを特徴とする請求項１７に記載の方法。

【請求項１９】
コンピュータシステムのプロセッサから前記コンピュータシステムのホストオペレーティングシステムを論理的に切り離しするための方法であって、
前記ゲストオペレーティングシステムの機能に関連するプロセッサ設定は、前記ゲストオペレーティングシステムのページテーブルに対するポインタであることを特徴とする請求項１７に記載の方法。

【請求項２０】
コンピュータシステムのプロセッサから前記コンピュータシステムのホストオペレーティングシステムを論理的に切り離しするための方法であって、
前記エミュレータプログラムによって要求された機能を実施するために前記ホストオペレーティングシステムを前記プロセッサに一時的に論理的に再結合するステップと、
前記ホストオペレーティングシステムが前記エミュレータプログラムによって要求された機能を完了した後、前記ゲストオペレーティングシステムに関連する前記プロセッサ設定のために前記エミュレータプログラムを前記プロセッサに論理的に再結合するステップとをさらに含むことを特徴とする請求項１７に記載の方法。

【請求項２１】
コンピュータシステムのプロセッサから前記コンピュータシステムのホストオペレーティングシステムを論理的に切り離しするための方法であって、第１の交換ステップは、
前記プロセッサの識別可能なレジスタから前記ホストオペレーティングシステムに関連するプロセッサ設定を読み取るステップと、
メモリからメモリに読み取られた前記プロセッサ設定を書き込むステップと
を含むことを特徴とする請求項１７に記載の方法。

【請求項２２】
コンピュータシステムのプロセッサから前記コンピュータシステムのホストオペレーティングシステムを論理的に切り離しするための方法であって、第２の交換ステップは、
メモリから前記ホストオペレーティングシステムに関連する前記プロセッサ設定を読み取るステップと、
前記プロセッサ設定を前記プロセッサの前記識別可能なレジスタに書き込むステップと
を含むことを特徴とする請求項２１に記載の方法。

【請求項２３】
コンピュータシステムのプロセッサから前記コンピュータシステムのホストオペレーティングシステムを論理的に切り離しするための方法であって、前記メモリはＲＡＭメモリで

あることを特徴とする請求項２１に記載の方法。
【請求項２４】
コンピュータシステムのプロセッサから前記コンピュータシステムのホストオペレーティングシステムを論理的に切り離しするための方法であって、前記ゲストオペレーティングシステムの機能に関連する前記プロセッサ設定は、前記ゲストオペレーティングシステムによって提供される例外ハンドラに対するポインタであることを特徴とする請求項１７に記載の方法。
【請求項２５】
コンピュータシステムのプロセッサによる呼び出しを、前記コンピュータシステムのホストオペレーティングシステム上でエミュレートされているゲストオペレーティングシステムに再ルーティングするための方法であって、
前記プロセッサから第１の組のプロセッサ設定を読み取るステップを含み、前記プロセッサ設定は前記ホストオペレーティングシステムによって提供される機能に関連し、
さらに、前記第１のプロセッサ設定をメモリ内に格納するステップと、
第２の組の交換プロセッサ設定を前記プロセッサに書き込むステップと
を含み、前記交換の組のプロセッサ設定は前記ゲストオペレーティングシステムによって提供される機能に関連し、前記交換組のプロセッサ設定に関連する機能に関して、前記ホストオペレーティングシステムは前記プロセッサから論理的に切り離しし、前記ゲストオペレーティングシステムは前記プロセッサに論理的に結合することを特徴とする方法。
【請求項２６】
コンピュータシステムのプロセッサによる呼び出しを、前記コンピュータシステムのホストオペレーティングシステム上でエミュレートされているゲストオペレーティングシステムに再ルーティングするための方法であって、
メモリ内に格納された前記第１の組のプロセッサ設定を読み取るステップと、
前記第１の組のプロセッサ設定を前記プロセッサに書き込むステップと
を含み、前記第１の組のプロセッサ設定に関連する機能に関して、前記ホストオペレーティングシステムは前記プロセッサに論理的に結合することを特徴とする請求項２５に記載の方法。
【請求項２７】
コンピュータシステムのプロセッサからホストオペレーティングシステムを論理的に切り離しするための方法であって、
前記プロセッサ上の１組のレジスタのコンテンツをメモリに保存するステップを含み、前記１組のレジスタは前記プロセッサの前記プロセッサ設定のうちの少なくとも一部を格納し、
さらに、前記１組のレジスタに、前記ホストオペレーティングシステム上でアプリケーションプログラムとして常駐するゲストオペレーティングシステムによって提供される機能に関連する交換組のプロセッサ設定を書き込むステップを含むことを特徴とする方法。
【請求項２８】
コンピュータシステムのプロセッサからホストオペレーティングシステムを論理的に切り離しするための方法であって、前記交換組のレジスタは、前記ホストオペレーティングシステムによって提供される例外ハンドラルーチンに対するポインタを有することを特徴とする請求項２０に記載の方法。
【請求項２９】
コンピュータシステムのプロセッサからホストオペレーティングシステムを論理的に切り離しするための方法であって、前記交換組のレジスタは、前記ホストオペレーティングシステムによって提供される割込ハンドラルーチンに対するポインタを有することを特徴とする請求項２７に記載の方法。
【請求項３０】
コンピュータシステムのプロセッサからホストオペレーティングシステムを論理的に切り離しするための方法であって、前記交換組のレジスタは、前記ホストオペレーティングシ

ステムに関連するページテーブルを有することを特徴とする請求項２７に記載の方法。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、一般に、コンピュータシステムのエミュレーションの分野に関し、より詳細には、ゲストオペレーティングシステムによるプロセッサ制御を支援するためにコンピュータシステムのプロセッサからホストオペレーティングシステムをトランスペアレント（透過的）に切り離しするための方法に関する。

【背景技術】

【０００２】

コンピュータは、特定のシステム命令セットを実行するように設計されている汎用中央処理ユニット（ＣＰＵ）を含んでいる。同様のアーキテクチャまたは設計仕様を有する１群のプロセッサを、同じプロセッサファミリのメンバであると考えることができる。現在のプロセッサファミリの例には、アリゾナ州フェニックス市（Ｐｈｏｅｎｉｘ，Ａｒｉｚｏｎａ）にあるモトローラ社製のＭｏｔｏｒｏｌａ６８０Ｘ０プロセッサファミリ、カリフォルニア州サニーヴェール市（Ｓｕｎｎｙｖａｌｅ，Ｃａｌｉｆｏｒｎｉａ）にあるインテル社製のＩｎｔｅｌ８０Ｘ８６プロセッサファミリ、モトローラ社製で、カリフォルニア州キューパーティーノ市（Ｃｕｐｅｒｔｉｎｏ，Ｃａｌｉｆｏｒｎｉａ）にあるアップルコンピュータ社製のコンピュータに使用されているＰｏｗｅｒＰＣプロセッサファミリがある。アーキテクチャおよび設計考慮事項が同様であることから１群のプロセッサは」、同じファミリであると考えられるが、そのクロック速度および他の性能パラメータによりそれらプロセッサは同じファミリ内でもかなり異なっていることがある。

【０００３】

それぞれのファミリのマイクロプロセッサが、そのプロセッサファミリにユニークである命令を実行する。１つのプロセッサまたは１ファミリのプロセッサが実行できる集合的な命令セットが、そのプロセッサの命令セットとして知られている。一例として、Ｉｎｔｅｌ８０Ｘ８６プロセッサファミリによって使用される命令セットは、ＰｏｗｅｒＰＣプロセッサファミリによって使用される命令セットとの互換性がない。Ｉｎｔｅｌ８０Ｘ８６命令セットは、複雑命令セットコンピュータ（ＣＩＳＣ）フォーマットに基づく。モトローラＰｏｗｅｒＰＣ命令セットは、縮小命令セットコンピュータ（ＲＩＳＣ）フォーマットに基づく。ＣＩＳＣプロセッサは多数の命令を使用し、そのうちのいくつかはかなり複雑な機能を実行できるが、一般に、実行するためには多くのクロックサイクルを必要とする。ＲＩＳＣプロセッサは、より少ない数の使用可能な命令を使用して、より単純な機能セットを実行する。この機能セットははるかに高い速度で実行される。

【０００４】

コンピュータシステム間のプロセッサファミリのユニーク性の結果としてまた、一般に、コンピュータシステムのハードウェアアーキテクチャの他の要素間に非互換性が生じる。Ｉｎｔｅｌ８０Ｘ８６プロセッサファミリのプロセッサとともに製造されるコンピュータシステムは、ＰｏｗｅｒＰＣプロセッサファミリのプロセッサとともに製造されるコンピュータシステムのハードウェアアーキテクチャとは異なるハードウェアアーキテクチャを有する。プロセッサ命令セットおよびコンピュータシステムのハードウェアアーキテクチャのユニーク性のために、アプリケーションソフトウェアプログラムは、一般に、特定のオペレーティングシステムを実行する特定のコンピュータシステム上を動作するよう書かれる。

【０００５】

コンピュータ製造業者は、そのコンピュータ製造業者の製品ラインに関連するマイクロプロセッサファミリ上でできるだけ多くの数のアプリケーションを動作させるようにすることにより自社の市場占有率を最大にしたいと望む。コンピュータシステム上で動作できるオペレーティングシステムおよびアプリケーションプログラムの数を拡張するために、ホストと呼ばれる、１タイプのＣＰＵを備えた所与のコンピュータが、エミュレータプログ

ラムを含むようにするという技術の分野が発展してきた。このエミュレータプログラムによりホストコンピュータがゲストと呼ばれる無関係のタイプのＣＰＵの命令をエミュレートすることができる。したがって、ホストコンピュータは、１つまたは複数のホスト命令を所与のゲスト命令に応答して呼び出すようにするアプリケーションを実行する。したがって、ホストコンピュータは、それ自体のハードウェアアーキテクチャのために設計されたソフトウェアおよび無関係のハードウェアアーキテクチャを有するコンピュータのために書かれたソフトウェアの双方を実行することができる。より具体的な例として、アップルコンピュータ製のコンピュータシステムが、たとえば、ＰＣベースのコンピュータシステムのために書かれたオペレーティングシステムおよびプログラムを実行することができる。エミュレータプログラムを使用して、単一のＣＰＵ上で多数の互換性のないオペレーティングシステムを同時に操作（ｏｐｅｒａｔｅ）することも可能である。この構成においては、それぞれのオペレーティングシステムは他のオペレーティングシステムとの互換性がないが、エミュレータプログラムが、２つのオペレーティングシステムのうちの１つのホストとして働くことができ、それにより、通常互換性のないオペレーティングシステムが同じコンピュータシステム上で同時に動作できるようになる。

【０００６】

ゲストコンピュータシステムがホストコンピュータシステム上でエミュレートされる場合は、ホストコンピュータシステムがゲストコンピュータシステムのハードウェアアーキテクチャのオペレーションのソフトウェアの実行としてのみ存在するので、ゲストコンピュータシステムは仮想マシンと言える。エミュレータおよび仮想マシンという用語は、時に、コンピュータシステム全体のハードウェアアーキテクチャを真似るかまたはエミュレートする能力を表すことと交換可能に使用される。一例として、カリフォルニア州サンマテオ市（Ｓａｎ　Ｍａｔｅｏ，Ｃａｌｉｆｏｒｎｉａ）にあるＣｏｎｎｅｃｔｉｘ社で作成された仮想ＰＣソフトウェアは、Ｉｎｔｅｌ８０Ｘ８６Ｐｅｎｔｉｕｍ（登録商標）プロセッサやさまざまなマザーボード構成要素およびカードを含むコンピュータ全体をエミュレートする。これらの構成要素のオペレーションは、ホストマシン上で動作している仮想マシン内でエミュレートされる。ＰｏｗｅｒＰＣプロセッサを備えたコンピュータシステムなどのホストコンピュータのオペレーティングシステムソフトウェアおよびハードウェアアーキテクチャ上で実行するエミュレータプログラムが、ゲストコンピュータシステム全体のオペレーションを真似る。エミュレータプログラムは、ホストマシンのハードウェアアーキテクチャと、エミュレートされる環境内で動作するソフトウェアによって伝送される命令との間のやりとりの機能を果たす。

【０００７】

ハイパーバイザは、ホストオペレーティングシステムのカーネルレベルの近くに存在する制御プログラムであり、ホストオペレーティングシステム以外の１つまたは複数の２次オペレーティングシステムが、コンピュータシステムのプロセッサを含むコンピュータシステムのハードウェアを使用できるよう動作する。オペレーティングシステムのハイパーバイザは、２次オペレーティングシステムのオペレーティング環境をエミュレートする。その結果、実際には別のハードウェアおよび／またはオペレーティングシステム環境内で動作している可能性があり、ホストオペレーティングシステムはコンピュータシステムの論理制御内である可能性がある場合に、２次オペレーティングシステムが、いつものハードウェアおよび／またはオペレーティングシステム環境内で動作しておりコンピュータシステムの論理制御内にあると信じる。多くのオペレーティングシステムは、オペレーティングシステムがあたかもコンピュータシステムのハードウェアの排他的論理制御内にあるかのように動作しなければならないように機能する。多数のオペレーティングシステムが単一のコンピュータシステム上で同時に機能するためには、それぞれのオペレーティングシステムのハイパーバイザは、それぞれのオペレーティングシステムがあたかもコンピュータシステム全体にわたって排他的制御を有するかのように機能するように、他のオペレーティングシステムの存在をマスクするように機能しなければならない。

【０００８】

ゲストシステムのハードウェアアーキテクチャ全体をエミュレートしたい場合は、たとえば、メモリ管理ユニット、コンテキストスイッチハードウェア、例外処理ベクトル、割込ルーチンポインタ、ホストプロセッサの浮動小数点ユニット、ホストコンピュータシステムのメモリを含む、ホストコンピュータシステム内のハードウェアを最大限に利用することが、しばしば有用であり望ましい。しかし、このハードウェアは、通常ホストオペレーティングシステムの制御下にあり、ユーザレベルまたはアプリケーションソフトウェアによるアクセスからは遮蔽されている。例外は、プロセッサによって解決できないコンピュータシステム内で起きるイベントである。割込みとは、他のいくつかのオペレーションを実行するために、現在の命令ストリームに一時的に割り込むためにプロセッサに送信される要求である。

【０００９】

このような困難な点を回避するための１つのオプションが、ホストコンピュータシステムのより低いレベルのハードウェアを使用することを試みないエミュレーションプログラムを用いることである。アプリケーションプログラムとして常駐しホストオペレーティングシステム上で動作するエミュレーションプログラムの場合に、この種のアプローチがしばしば使用される。これらのアプリケーションエミュレーションプログラムは、もちろん、ホストコンピュータシステムのホストオペレーティングシステム上で動作するアプリケーションプログラムであるという性能上の不利益を受けやすい。さらに、これらのアプリケーションエミュレーションプログラムは、ハードウェア機能（ｈａｒｄｗａｒｅ ｆｅａｔｕｒｅｓ）へのアクセスを得るだけであり、これらのハードウェア機能は、ホストオペレーティングシステムによりアプリケーションエミュレーションプログラムに対しさらされる。一例として、ホストオペレーティングシステム上でアプリケーションとして動作するエミュレーションプログラムが、例外ハンドラを含むことができる。困難な点は、エミュレーションプログラムの例外ハンドラが、いくつかのソフトウェアレイヤによりホストコンピュータシステムのプロセッサから分離されており、そのため、ホストプロセッサが例外を発行する時間と例外がエミュレーションプログラムの例外ハンドラに識別される時間の間に待ち時間が生まれることである。別のオプションは、エミュレーションプログラムによるコンピュータシステムの共有のためのサポートをホストオペレーティングシステムハイパーバイザに書き込むことができるとみなしてエミュレーションプログラムがコンピュータシステムとハードウェアを共用することである。

【００１０】

エミュレートされるコンピューティング環境内でハイパーバイザを使用する場合の別の困難な点が、ハイパーバイザ自体のメモリ空間とともにエミュレートされるコンピュータシステムのメモリ空間を管理することである。エミュレートされるコンピュータシステムのオペレーションを改良するために、ハイパーバイザは一般に、ホストコンピュータシステムのページテーブルがゲストオペレーティングシステムのページテーブルを大部分ミラーリングできるように、ホストコンピュータシステムのページテーブルを操作しようとする。このアプローチの１つの困難な点が、ハイパーバイザコードおよびそれに関連するデータが、ゲストオペレーティングシステムのページテーブルにマッピングされないことである。エミュレートされるコンピュータシステムが、ハイパーバイザがホストコンピュータシステムのハードウェア上でシステムのホストとして働いていることを認識しないため、必ずこのようになる。したがって、ハイパーバイザのページテーブルは、ホストコンピュータシステムのページテーブルと完全には合致しない。ハイパーバイザのページテーブルは、ハイパーバイザ自体のコードおよびデータのマッピングを含んでいる。

【発明の開示】

【発明が解決しようとする課題】

【００１１】

ゲストオペレーティングシステムのアドレスマッピングが、ハイパーバイザ自体のコードおよびデータのマッピングされた場所全体に対するアドレスのマッピングを含むことがあるので、ハイパーバイザによるゲストオペレーティングシステムのページテーブルのミラ

ーリングは、ゲストオペレーティングシステムのアドレスマッピングがハイパーバイザのアドレスマッピングと競合する可能性を上げる。ページマッピングの競合というこの問題の１つのソリューションが、ゲストオペレーティングシステムの論理メモリのマッピングが、ハイパーバイザのほぼ同一のページテーブルによって規定されたマッピングと競合する場合に、ゲストオペレーティングシステムのページテーブルを変更することである。しかし、エミュレートされるコンピュータシステムのページテーブルを変更することは、エミュレートされるコンピュータシステムにそのページテーブルへの予期せぬ変更を導入する可能性があり、恐らくゲストオペレーティングシステム内の他の非互換性や障害を導入するという点で望ましくない。

【課題を解決するための手段】

【００１２】

本発明は、コンピュータシステムのプロセッサからホストオペレーティングシステムを論理的に切り離しする（ｄｅｃｏｕｐｌｉｎｇ）ための技術に係わる。本発明の方法によれば、エミュレーションプログラムのハイパーバイザが、プロセッサの１組のプロセッサ設定（ｓｅｔｔｉｎｇ）を読み取る。これらのプロセッサ設定は、ハイパーバイザによって主記憶装置に保存される。ハイパーバイザは、これらのプロセッサ設定をゲストオペレーティングシステムによって提供されるソフトウェアルーチンまたはデータ構造に関連する１組のプロセッサ設定に交換する。これらのソフトウェアルーチンまたはデータ構造は、例外および割込ハンドラルーチンとページテーブルとを有することができる。プロセッサ設定の置換の結果として、ホストオペレーティングシステムは、プロセッサのプロセッサ設定のうちの少なくともいくつかがゲストオペレーティングシステムによって提供されるソフトウェアルーチンまたはデータ構造に関連し、ホストオペレーティングシステムによって提供されるソフトウェアルーチンまたはデータ構造にもはや関連しないという点で、プロセッサ制御から切り離しされている。ホストオペレーティングシステムをプロセッサに再結合させるためには、ハイパーバイザは、保存されたプロセッサ設定をメモリから読み取り、これらのプロセッサ設定をプロセッサの適切なレジスタに書き込む。

【００１３】

ハイパーバイザがプロセッサに結合している期間中は、ハイパーバイザのページテーブルはゲストオペレーティングシステムのページテーブルを大部分ミラーリングするが、ハイパーバイザのページテーブルはハイパーバイザコード自体のマッピングされた場所を含むという主要相違点を有する。ゲストオペレーティングシステムのページテーブルとハイパーバイザのページテーブルの間の競合を回避するためには、ゲストオペレーティングシステムのページテーブルへの変更の結果、ゲストオペレーティングシステム内のマッピングされたメモリ場所が、ハイパーバイザのページテーブル内のハイパーバイザコードのマッピングされた場所にオーバラップする場合に、ハイパーバイザのページテーブルが再割り当てされる。

【００１４】

本明細書に開示されている論理的に切り離し再結合する方法は、ゲストオペレーティングシステムによって提供できるいつくかの機能について、ゲストオペレーティングシステムをプロセッサとより厳密に調整できるようにするという点で有利である。この方式により、例外ハンドラ呼び出し、割込ハンドラ呼び出し、メモリ管理呼び出しが、ゲストオペレーティングシステムに渡される前に、プロセッサからホストオペレーティングシステムに渡される必要がない。むしろ、プロセッサからのこれらの呼び出しは、エミュレーションプログラムのハイパーバイザを介してゲストオペレーティングシステムに直接に渡される。

【００１５】

本明細書に開示されている方法はまた、ハイパーバイザまたはエミュレーションプログラムの切り離しツールが、ホストオペレーティングシステムのソースコードへのアクセスを有する必要がないという点でも有利である。むしろ、ハイパーバイザは、ホストオペレーティングシステムに対してトランスペアレントな方式で、プロセッサからホストオペレー

ティングシステムを切り離し、それがプロセッサから切り離しまたはそれに再結合していることを認識しない。

【００１６】

本明細書に開示されている方法はまた、たとえホストオペレーティングシステムが、通常ホストオペレーティングシステムによって提供されるいくつかの機能について、プロセッサから切り離ししている場合でも、ホストオペレーティングシステムは、ホストプロセッサの機能の残りの部分については、ホストプロセッサに結合したままであるという点でも有利である。この期間中は、ホストオペレーティングシステムは、引き続きコンピュータシステムのハードウェア機能の多くを管理し、それにより、ゲストオペレーティングシステムがホストオペレーティングシステムを通じてプリント要求をルーティングできるようになる。

【００１７】

ゲストコンピュータシステムとの競合が起きた場合にハイパーバイザのページテーブルを再割り当てすることの利点は、ページテーブルの再割当てにより、ハイパーバイザのページテーブルがゲストオペレーティングシステムのページテーブルを覆い、同時にハイパーバイザ自体のコードをマッピングできるようになることである。この環境においては、ゲストオペレーティングシステムのページテーブルへの変更の結果として、ゲストオペレーティングシステムのページテーブルとハイパーバイザのページテーブルの間の競合が生じない。さらに、ハイパーバイザのページテーブルの再割当ては、ゲストオペレーティングシステムのオペレーションに対してトランスペアレントであり、それにより、ゲストオペレーティングシステム内でシステムレベルの競合が作り出される可能性が回避される。

【００１８】

本発明の他の技術的な利点は、以下の図、説明、特許請求の範囲から当業者には容易に自明となろう。

【発明を実施するための最良の形態】

【００１９】

本発明およびその利点のより完全な理解については、以下の説明を添付図面と合わせて参照することにより得られるであろう。ここで同様の参照番号は、同様の機能を示す。

【００２０】

本発明は、エミュレーションソフトウェアが、ホストコンピュータシステムのより低いレベルのハードウェア構成要素にアクセスできるようにするエミュレーション環境を提供する。本発明は、ホストオペレーティングシステムに対してトランスペアレントであるハイパーバイザを含むエミュレーションプログラムを含む。エミュレーションプログラムのハイパーバイザは、ホストコンピュータシステムの排他的制御からオペレーティングシステムを、短時間、論理的に切断するかまたは切り離し、この時間中、エミュレーションプログラムのハイパーバイザは、ホストコンピュータシステムのハードウェアの制御内に置かれ、それにより、エミュレーションプログラムが、短時間、ホストコンピュータシステムの一定のハードウェア機能に論理的に近くなることができる。

【００２１】

プロセッサのプロセッサ状態は、ある単一の時のプロセッサ設定のすべてのスナップショットである。これらの設定は、プロセッサが操作しているデータとプロセッサによって使用されるプログラムカウンタ、ポインタ、他の操作可能なフラグとを有する。データがプロセッサによって操作され、多くの他のプロセッサ設定がそれぞれのプロセッササイクルで変更することがあるので、プロセッサ状態も同様に、それぞれのプロセッササイクルで変更することがあり、毎秒何百万回も変更することがある。

【００２２】

1サブセットのプロセッサ状態がユーザ状態である。プロセッサによって操作されるデータおよびプロセッサのプログラムカウンタは、まとまってプロセッサのユーザ状態を有する。ユーザ状態は、ユーザ状態を有する値がアプリケーションレベルのプログラムによって操作できるためにその名が付けられている。プロセッサ状態のプロセッサ設定の残りの

部分は、プロセッサのスーパバイザ状態または特権状態を有する。スーパバイザ状態の値は、コンピュータシステムのオペレーティングシステムの一部であるソフトウェアによってのみ修正できる。プロセッサ状態のスーパバイザ状態または特権状態の設定は、アプリケーションレベルのプログラムからアクセスできない。アプリケーションレベルのプログラムからアクセス可能なユーザ状態とアプリケーションレベルのプログラムからアクセス可能でないスーパバイザ状態の間のプロセッサ設定の分離により、たとえアプリケーションレベルのプログラムがクラッシュしたり決定的エラーを引き起こした場合でも、オペレーティングシステムは機能し続けることができる。

【００２３】

プロセッサのスーパバイザ状態の設定は、一般に、いくつかのカテゴリのうちの１つに当てはまる。プロセッサのスーパバイザ状態の１つのカテゴリが、プロセッサのモードフラグであり、プロセッサに、一定の計算状況、例外挙動、レガシプログラムのサポート、またはプロセッサのより新しい機能の選択的使用をサポートするためのいくつかのモードの１つに従って動作するよう命令する設定を有する。プロセッサ状態の設定の別のカテゴリが、外部ハードウェアアクセスを制御する設定である。プロセッサが、しばしば、プロセッサと、キャッシュ、メモリ、他の入出力デバイスなどのプロセッサの外部のハードウェアの間の通信のためのいくつかの操作可能な設定をサポートする。たとえば、通信速度や帯域幅を含むこれらの操作可能な設定が、特権レベルまたはスーパバイザレベルのコードによってのみ修正できるので、それらは、プロセッサのスーパバイザ状態の一部と考えられる。

【００２４】

プロセッサの特権状態の設定の別のカテゴリが、例外および割込ルーチンポインタである。プロセッサが例外に遭遇したりまたは割込みを求められた場合は、プロセッサは、現在のプログラムカウンタアドレスで実行を停止し、所定の例外ルーチンアドレスで実行を開始する。これらの例外ルーチンの場所は、例外ルーチンポインタの使用を通じて識別される。いくつかのプロセッサについては、例外ルーチンは、ハードコードされたメモリアドレスに置かれる。しかし、ほとんどのプロセッサ上では、それらの場所はプログラム可能であり、それらの場所の値は、プロセッサのスーパバイザ状態の一部であると考えられる。

【００２５】

１組のデータ構造ポインタが、プロセッサの特権状態の別のカテゴリを構成する。プロセッサ状態のいくつかのデータ構造は大きすぎてプロセッサ内に格納できないので、これらのデータ構造は主記憶装置内に格納され、データ構造に対するポインタがプロセッサに格納される。この種の大きなデータ構造の一例が、プロセッサのページテーブルである。これらのテーブルが、プロセッサのために、物理メモリ場所への仮想メモリアクセスの変換に必要なメモリアドレスの論理から物理への翻訳を定義する。プロセッサに格納される、ページテーブルとページテーブルに対するポインタの両方が、プロセッサのスーパバイザ状態の一部であると考えられる。主記憶装置内に格納されるプロセッサのスーパバイザ状態のデータ構造の別の例が、例外ルーチンポインタテーブルである。プロセッサにそれぞれの例外ルーチンのためのポインタを格納するよりはむしろ、プロセッサに格納された単一のポインタが、例外ルーチンポインタのテーブルを指し示すことができる。例外ポインタまたはベクトルテーブルは、プロセッサからアクセス可能な例外ハンドラに対するアドレスポインタのリスティングを有する。プロセッサに格納される例外ルーチンポインタと例外ルーチンポインタテーブルの両方が、スーパバイザ状態の一部であると考えられる。

【００２６】

ホストオペレーティングシステムは、一般に、プロセッサのために、例外および割込ハンドラルーチンに対するベクトルまたはポインタを確立する。たとえば、プロセッサが、プロセッサにゼロによる除算をさせる１組のコマンドを処理するよう命令された場合に、プロセッサは、ゼロによる除算の例外を扱う例外ハンドラルーチンを呼び出す。そうすることにより、プロセッサは、プロセッサに格納されたベクトルによって指し示された例外ハ

ンドラルーチンを呼び出すことができる。代わりに、プロセッサは、例外条件のための正しい例外ハンドラのための例外ベクトルテーブルをスキャンし、次いで、適用可能な例外ハンドラに関連する例外ベクトルまたはポインタを使用して例外ハンドラを呼び出すことができる。

【００２７】

ホストオペレーティングシステムがホストプロセッサの論理制御内にある場合は、ホストオペレーティングシステムは、ホストプロセッサのために、ホストプロセッサが必要としそれによってサポートされている例外ベクトルおよび他のすべてのベクトルを定義する。プロセッサが例外条件に遭遇したり、またはページテーブルルーチンやページテーブルベクトルなどのベクトルによって参照される別のルーチンにアクセスする必要がある場合には、プロセッサは、ホストオペレーティングシステムの例外ハンドラを実行する。

【００２８】

エミュレートされるコンピュータシステムの場合は、エミュレーションプログラムが、ホストコンピュータシステム内にエミュレートされるオペレーティング環境を提供する。図１は、コンピュータシステム１０内のエミュレートされるオペレーティング環境のためのハードウェアおよびソフトウェアアーキテクチャの論理レイヤを示す図である。エミュレーションプログラム１４が、ホストコンピュータシステムハードウェアまたはプロセッサ１１上で実行するホストオペレーティングシステム上を動作する。エミュレーションプログラム１４が、ゲストオペレーティングシステム１８を含むゲストコンピュータシステム１６をエミュレートする。ゲストアプリケーションプログラムが、ゲストオペレーティングシステム１８上で実行できる。図１のエミュレートされるオペレーティング環境においては、エミュレーションプログラム１４のオペレーションのために、たとえゲストアプリケーション２０が一般にホストオペレーティングシステム１２およびホストコンピュータシステムハードウェア１１とは互換性のないオペレーティングシステム上を動作するよう設計されていたとしても、ゲストアプリケーション２０はコンピュータシステム１０上を動作できる。図１のアーキテクチャにおいては、ゲストオペレーティングシステム１８が、ホストオペレーティングシステム１２を含む、いくつかの論理およびソフトウェアレイヤを介してホストコンピュータシステムハードウェア１１から分離されている。この論理的分離により、ゲストオペレーティングシステムがホストコンピュータシステムハードウェア１１と直接に通信することを試みる場合に、待ち時間および性能上の困難が導入される。

【００２９】

図１の例では、ホストオペレーティングシステム１２が、ホストコンピュータシステムのプロセッサを含むホストコンピュータシステムハードウェア１１の排他的動作制御内にある。ホストオペレーティングシステムは、ホストコンピュータシステムのプロセッサのために、プロセッサ状態のいくつかの設定を確立している。たとえば、ホストオペレーティングシステム１２は、ホストプロセッサのベクトルテーブル内またはレジスタ内に置かれている１組の例外ハンドラベクトル、１組の割込ハンドラ、ページテーブルベクトルを確立した可能性がある。したがって、プロセッサが例外ハンドラ呼び出し割込ハンドラ呼び出し、またはメモリ管理呼び出しをした場合は、これらの呼び出しのためのプロセッサ設定が、ホストオペレーティングシステムによって排他的ベースで提供される。ホストオペレーティングシステムが、プロセッサのスーパバイザまたは特権状態の設定を確立したので、ホストオペレーティングシステムは、ホストコンピュータシステムのプロセッサの論理制御内にあると言える。

【００３０】

本発明のエミュレーション技術によれば、エミュレーションプログラム自体は、プロセッサおよび他のホストコンピュータシステムハードウェア１１の排他的制御からホストオペレーティングシステムを論理的に切断するかまたは切り離しすることのできるハイパーバイザを有する。エミュレーションプログラム１４のハイパーバイザは、ホストプロセッサによって使用されるベクトルおよび他のアドレスポインタを読み取り、それをメモリ場所

内に保存することにより論理切断ステップを達成する。ホストコンピュータシステムの排他的制御からホストオペレーティングシステムを論理的に切断し、エミュレーションプログラムのハイパーバイザをホストコンピュータシステムに論理的に接続するかまたは結合するためにハイパーバイザによってとられるステップの流れ図が、図２に示されている。エミュレーションプログラムのハイパーバイザが、プロセッサのプロセッサ設定のうちの少なくともいくつかに対する制御を表明したい場合は、ハイパーバイザは、ステップ２４で、プロセッサのスーパバイザまたは特権状態の設定のうちのいくつかまたはすべてを読み取る。本発明の一実施形態によれば、プロセッサは、例外ベクトルまたは例外ベクトルテーブルに対するポインタ、割込ポインタまたは割込ルーチンテーブルに対するポインタ、ページテーブルに対するポインタを含む、プロセッサの適用可能なレジスタからアドレスポインタを読み取る。ステップ２６で、ハイパーバイザは、ホストプロセッサ上のそれらに関連する記憶装置場所の識別とともにこれらのプロセッサ設定をメモリに保存する。次いで、ハイパーバイザは、ステップ２８で、ゲストオペレーティングシステムのユーザレベルのプロセッサ状態およびエミュレーションプログラムのハイパーバイザの特権レベル状態を置換する。その結果、ホストプロセッサのプロセッサ設定は、ゲストオペレーティングシステムのユーザレベルのプロセッサ状態と、エミュレーションプログラムのハイパーバイザに関連する特権レベル状態との組み合わせと交換される。そうすることによりハイパーバイザは、プロセッサの適切なレジスタ内に、例外ハンドラに対するベクトルおよび他のアドレスポインタ、割込ルーチン、エミュレーションプログラムのハイパーバイザおよびゲストオペレーティングシステムのページマップを格納する。

【００３１】

エミュレーションプログラムのハイパーバイザによって置換されたプロセッサ設定が、スーパバイザまたは特権レベルの設定であるので、エミュレーションプログラムのハイパーバイザは、置換されたプロセッサ設定のすべてについてプロセッサの論理制御を有する。この方式により、エミュレートされるコンピュータシステムは、置換されたプロセッサ設定によって統治される機能について、論理的にホストコンピュータシステムのハードウェア機能の制御内にある。プロセッサ設定の置換後のエミュレーションプログラム１４の論理関係の図が、図３に示されている。図３のハイパーバイザ２２が、エミュレーションプログラム１４の１つの構成要素として示され、プロセッサおよびホストハードウェア１１への論理通信リンクを有するものとして示されている。

【００３２】

ホストハードウェア１１のエミュレーションプログラムの論理制御の一例として、例外がこの時間中に起きる場合は、プロセッサは、エミュレーションプログラムのハイパーバイザによって提供される例外ハンドラを呼び出す。同様に、プロセッサが、仮想メモリアドレスを物理メモリアドレスに翻訳するよう呼び出される場合は、プロセッサは、ゲストオペレーティングシステムに関連するページテーブルにアクセスする。したがって、プロセッサ状態の多くの設定が交換されて、その結果、プロセッサは、ゲストオペレーティングシステムによって直接提供されるソフトウェアまたはデータ構造を呼び出すかまたはそれにアクセスしている。そのことにより、ゲストオペレーティングシステムが、コンピュータシステムのプロセッサとより厳密に論理的に調整できるようになり、ホストオペレーティングシステム全体にわたってコンピュータシステムのプロセッサからゲストオペレーティングシステムを分離することにより引き起こされる待ち時間がなくなる。例外の場合は、ハイパーバイザの例外ハンドラは例外自体を扱うことができるか、またはハイパーバイザの例外ハンドラはゲストオペレーティングシステムの例外ハンドラを起動でき、それにより、ゲストオペレーティングシステムは、例外呼び出しをホストオペレーティングシステムソフトウェアレイヤを通じて通過させる必要がなく、例外を直接扱うことができるようになる。

【００３３】

図３に示されるように、ホストオペレーティングシステム１２は、ホストコンピュータシステムのハードウェアから永久に切断されているわけではない。１組のプロセッサ設定を

介してハイパーバイザによって表明された論理制御の結果、ハイパーバイザ２２が、コンピュータシステム内のすべてのハードウェアインタラクションに対する責任を負うことにはならない。ホストオペレーティングシステムがコンピュータシステムのハードウェアに関係する機能要求を処理している時間中は、ホストオペレーティングシステムは、コンピュータシステムの論理制御内でなければならない。このことを達成するためには、ホストオペレーティングシステムは、一時的にコンピュータシステムのハードウェアに論理的に再結合しなければならない。いったんホストオペレーティングシステムがエミュレーションプログラムによって要求された機能を完了すると、エミュレーションプログラムは、エミュレーションプログラムのプロセッサ設定に関連する例外および割込ハンドラに対するその論理制御を再開できる。

【００３４】

図２の流れ図に関して、ステップ３０で、ハイパーバイザ２２が、プロセッサ設定に対する論理制御をホストオペレーティングシステムに返す。ハイパーバイザ２２が、メモリに保存されているベクトルおよび他のポインタをメモリから検索する。これらのアドレスは、適切なレジスタ場所に再び書き込まれ、これらのプロセッサ設定に対する論理制御をホストオペレーティングシステムに返す。ホストオペレーティングシステムがプロセッサから論理的に切断されている時間中は、ホストオペレーティングシステム１２の論理切断は、ホストオペレーティングシステム１２にとって明らかではない。ホストオペレーティングシステム１２は、１組のプロセッサ設定が置換され、その結果、いくつかのハードウェア機能のための論理制御がエミュレーションプログラムへ転送されたことを認識していない。論理制御がホストオペレーティングシステム１２に返された場合は、ホストオペレーティングシステム１２は、同様に、１組のハードウェア呼び出しの論理制御がそれに返されたことも認識していない。

【００３５】

エミュレーションプログラムによる、ホストオペレーティングシステムを通じて濾過される、例外、割込み、メモリ管理機能の処理のオプションと比べて、プロセッサスーパバイザ状態のプロセッサ設定のうちのいくつかの置換の結果として、エミュレーションプログラムによる性能ゲインが生じる。エミュレーションプログラムが、ホストオペレーティングシステム上に常駐するアプリケーションプログラムとして動作する場合においては、プロセッサの機能に対する排他的制御からホストオペレーティングシステムを論理的に切断することは、エミュレートされるコンピュータシステムのための性能上の利点である。いったんホストオペレーティングシステムがプロセッサの機能のうちの少なくともいくつかから論理的に切り離しされると、エミュレーションプログラムは、ホストオペレーティングシステムのプロセッサ設定の代わりにそのプロセッサ設定を置換することにより、ホストプロセッサとはるかにより厳密に動作できる。いったんプロセッサ設定が置換されると、エミュレーションプログラムは、プロセッサにエミュレーションプログラムの別々の組の例外ベクトル、割込ルーチン、ページテーブルに指示でき、それにより、ホストオペレーティングシステムがエミュレーションプログラムからプロセッサを分離させた場合に起きる待ち時間の問題を回避できる。

【００３６】

本発明のハイパーバイザの割込みおよび例外処理機能により、ゲストオペレーティングシステムによって予想される例外および割込みの処理も可能となる。オペレーティングシステムが、これらの例外を例外によって開始されたアプリケーションに渡すことなく、オペレーティングシステムレベルで一定の例外を扱うことを選ぶことができる。一例が、ゼロによる除算の例外であり、これはしばしば、例外によって開始されたアプリケーションプログラムに渡されることなく、オペレーティングシステムによって扱われる。エミュレーションプログラムのハイパーバイザは、ハイパーバイザ内で例外ハンドラを開始する必要がなく、引き続き例外をゲストオペレーティングシステムに渡すことができる。したがって予想されるように、ゲストアプリケーションプログラムにより、置換されたプロセッサ設定に関してエミュレーションプログラムがプロセッサの論理制御内である時にゼロによ

る除算の例外が開始された場合に、エミュレーションプログラムのハイパーバイザは、例外をゲストオペレーティングシステムに渡すことができる。

【００３７】

図４は、ゲストオペレーティングシステムのページテーブルをマッピングするための方法を示す流れ図である。ステップ４０で、ハイパーバイザは、ゲストオペレーティングシステムのページテーブルへの書込みを識別する。ステップ４２で、ハイパーバイザは、ゲストオペレーティングシステムのページテーブルへの書込みの結果、ゲストオペレーティングシステムのページテーブルとハイパーバイザのページテーブルの間の競合をマッピングすることが生じるかどうか判断する。ハイパーバイザのページテーブルは、ゲストオペレーティングシステムのページテーブルを厳密にミラーリングするが、ハイパーバイザのページテーブルがハイパーバイザのコードおよびデータをそのページテーブルにマッピングするという主要相違点を有する。ハイパーバイザは、たとえば、ゲストオペレーティングシステムのページテーブル内のマッピングされたページが、ハイパーバイザ内のページテーブル内のハイパーバイザのマッピングされた場所にオーバラップしている場合に、競合があると判断する。

【００３８】

ハイパーバイザが、ゲストオペレーティングシステムのページテーブルへの変更によって引き起こされた競合がないと判断した場合は、ハイパーバイザのページテーブルへの変更は必要ではない（ステップ４４）。ハイパーバイザが、ゲストオペレーティングシステムのページテーブルへの変更により、ホストコンピュータシステムのページテーブルとの競合が起きると判断した場合は、ハイパーバイザは、ステップ４６でそのページテーブルを再割り当てし、その結果、ハイパーバイザ自体のコードおよびデータは、ハイパーバイザのページテーブル内の別の場所に移動するかまたはマッピングされる。ゲストオペレーティングシステムのページテーブルへの変更を識別し、必要ならば、それに応答してハイパーバイザのページテーブルを再割り当てするこのプロセスは、ゲストオペレーティングシステムのオペレーションに対してトランスペアレントである。必要な場合に、いったんハイパーバイザのページテーブルが再割り当てされると、ゲストオペレーティングシステムのページテーブルは、もはやハイパーバイザのページテーブルと競合しない。

【００３９】

本発明は、その適用が、特定のコンピュータシステムアーキテクチャ、特にＩｎｔｅｌ８０Ｘ８６アーキテクチャのエミュレーションに限定されるものではない。むしろ、本明細書に開示されているエミュレーション技術は、プロセッサのプロセッサ設定のうちの少なくともいくつかに関して、ホストオペレーティングシステムがプロセッサからトランスペアレントに切断されていることが望ましい場合はいつでも適用可能である。

【００４０】

本発明を詳細に説明してきたが、頭記の特許請求の範囲に定義した本発明の趣旨および範囲から逸脱することなく、本発明にさまざまな変更、代替、改変を行うことができることを理解されたい。

【図面の簡単な説明】

【００４１】

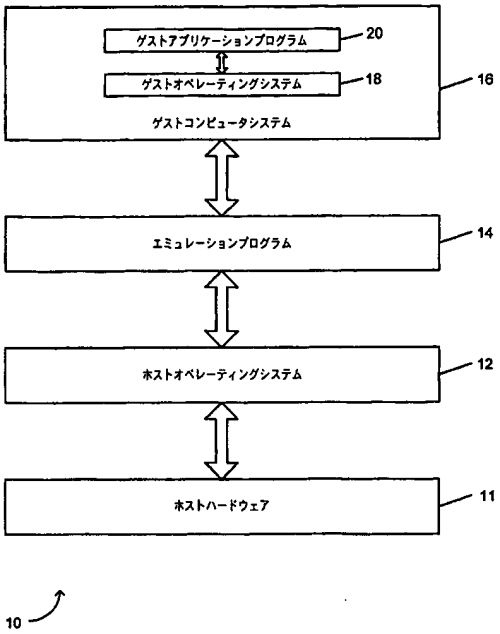
【図１】ホストコンピュータシステム内を動作するエミュレートされるコンピュータシステムの素子の論理関係を示す図である。

【図２】エミュレーションプログラムのハイパーバイザをコンピュータシステムのプロセッサに論理的に結合するための方法を示す流れ図である。

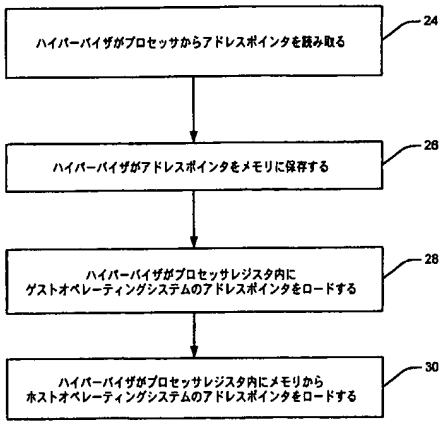
【図３】ハイパーバイザがコンピュータシステムのプロセッサに論理的に結合されているエミュレートされるコンピュータシステムの素子の論理関係を示す図である。

【図４】ゲストオペレーティングシステムのページテーブルをマッピングするための方法を示す流れ図である。

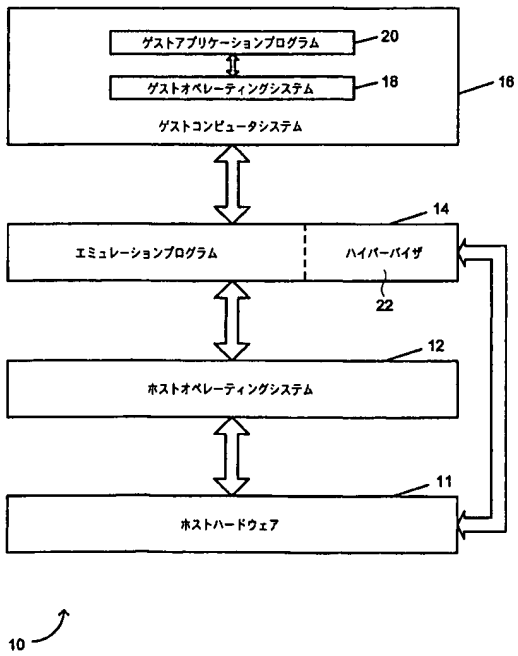
【図１】


ゲストアプリケーションプログラム
20
ゲストオペレーティングシステム
18
16
ゲストコンピュータシステム
エミュレーションプログラム
14
ホストオペレーティングシステム
12
ホストハードウェア
11
10

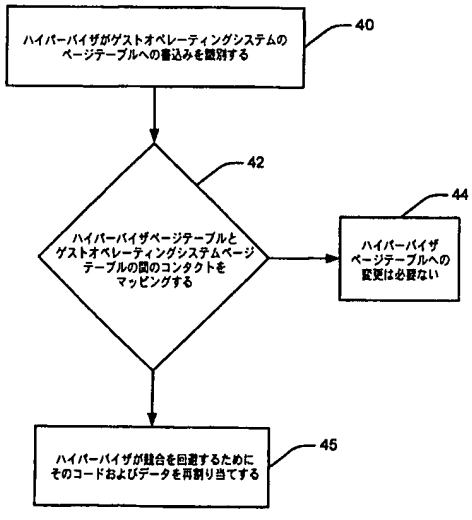
【図２】


ハイパーバイザがプロセッサからアドレスポインタを読み取る
24
ハイパーバイザがアドレスポインタをメモリに保存する
26
ハイパーバイザがプロセッサレジスタ内に
ゲストオペレーティングシステムのアドレスポインタをロードする
28
ハイパーバイザがプロセッサレジスタ内にメモリから
ホストオペレーティングシステムのアドレスポインタをロードする
30

【図３】


ゲストアプリケーションプログラム
20
ゲストオペレーティングシステム
18
16
ゲストコンピュータシステム
14
エミュレーションプログラム
ハイパーバイザ
22
12
ホストオペレーティングシステム
11
ホストハードウェア
10

【図４】


ハイパーバイザがゲストオペレーティングシステムの
ページテーブルへの書込みを識別する
40
42
ハイパーバイザページテーブルと
ゲストオペレーティングシステムページ
テーブルの間のコンタクトを
マッピングする
44
ハイパーバイザ
ページテーブルへの
変更は必要ない
ハイパーバイザが競合を回避するために
そのコードおよびデータを再割り当てする
45

【国際公開パンフレット】

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
27 June 2002 (27.06.2002)

PCT

(10) International Publication Number
WO 02/50664 A2

(51) International Patent Classification⁷: G06F 9/00

(21) International Application Number: PCT/US01/49217

(22) International Filing Date:
19 December 2001 (19.12.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/747,492 21 December 2000 (21.12.2000) US
09/906,392 16 July 2001 (16.07.2001) US

(71) Applicant: CONNECTIX CORPORATION [US/US]; 2955 Campus Drive, Suite 100, San Mateo, CA 94403 (US).

(72) Inventor: TRAUT, Eric, P.; 3 Iris Lane, San Carlos, CA 94070 (US).

(74) Agents: FULGHUM, Roger, J.; Baker Botts L.L.P., One Shell Plaza, 910 Louisiana, Houston, TX 77002 et al. (US).

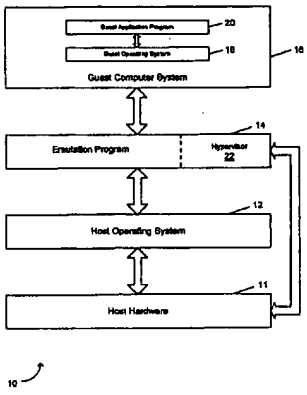
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

[Continued on next page]

(54) Title: LOGICAL SUBSTITUTION OF PROCESSOR CONTROL IN AN EMULATED COMPUTING ENVIRONMENT



(57) Abstract: In an emulated computing environment, a method is provided for logically decoupling the host operating system from the processor system with respect to certain processor settings of the processor. A hypervisor of the emulation program replaces some of the processor settings of the processor with processor settings associated with software routines or data structures provided by the guest operating system; The replaced processor settings are written to memory. During this period, when the processor calls a software routine or accesses a data structure associated with the replaced processor setting, the processor will call or access a software routine or access a data structure associated with the guest operating system, bypassing the host operating system and communicating directly with the guest operating system. When the host operating system is to be recoupled to the processor, the processor settings that have been saved to memory are rewritten to the appropriate registers of the processor. During the period that the hypervisor is coupled to the processor, the page table of the hypervisor is established such that it largely mirrors the page table of the guest operating system. If a change to the page table of the guest operating system causes a conflict between the page table of the guest operating system and the page table of the hypervisor, the page table of the hypervisor is reallocated in a manner that is transparent to the guest operating system.

WO 02/50664 A2

WO 02/50664 A2

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

SYSTEM AND METHOD FOR THE LOGICAL SUBSTITUTION OF PROCESSOR CONTROL IN AN EMULATED COMPUTING ENVIRONMENT

TECHNICAL FIELD OF THE INVENTION

The present invention relates in general to the field of computer system emulation and, more particularly, to a method for transparently decoupling the host operating system from the processor of the computer system in favor of control by the processor by the guest operating system.

BACKGROUND OF THE INVENTION

Computers include general purpose central processing units (CPUs) that are designed to execute a specific set of system instructions. A group of processors that have similar architecture or design specifications may be considered to be members of the same processor family. Examples of current processor families include the Motorola 680X0 processor family, manufactured by Motorola, Inc. of Phoenix, Arizona; the Intel 80X86 processor family, manufactured by Intel Corporation of Sunnyvale, California; and the PowerPC processor family, which is manufactured by Motorola, Inc. and used in computers manufactured by Apple Computer, Inc. of Cupertino, California. Although a group of processors may be in the same family because of their similar architecture and design considerations, processors may vary widely within a family according to their clock speed and other performance parameters.

Each family of microprocessors executes instructions that are unique to the processor family. The collective set of instructions that a processor or family of processors can execute is known as the processor's instruction set. As an example, the instruction set used by the Intel 80X86 processor family is incompatible with the instruction set used by the PowerPC processor family. The Intel 80X86 instruction set is based on the Complex Instruction Set Computer (CISC) format. The Motorola PowerPC instruction set is based on the Reduced Instruction Set Computer (RISC) format. CISC processors use a large number of instructions, some of which can perform rather complicated functions, but which require generally many clock cycles to execute. RISC processors use a smaller number of available instructions to perform a simpler set of functions that are executed at a much higher rate.

The uniqueness of the processor family among computer systems also typically results in incompatibility among the other elements of hardware architecture of the computer systems. A computer system manufactured with a processor from the Intel 80X86 processor family will have a hardware architecture that is different from the hardware architecture of a computer system manufactured with a processor from the PowerPC processor family. Because of the uniqueness of the processor instruction set and a computer system's hardware architecture, application software programs are typically written to run on a particular computer system running a particular operating system.

A computer manufacturer will want to maximize its market share by having more rather than fewer applications run on the microprocessor family associated with the computer manufacturer's product line. To expand the number of operating systems and application programs that can run on a computer system, a field of technology has developed in which a given computer having one type of CPU, called a host, will include an emulator program that allows the host computer to emulate the instructions of an unrelated type of CPU, called a guest. Thus, the host computer will execute an application that will cause one or more host instructions to be called in response to a given guest instruction. Thus, the host computer can both run software designed for its own hardware architecture and software written for computers having an unrelated hardware architecture. As a more specific example, a computer system manufactured by Apple Computer, for example, may run operating systems and programs written for PC-based computer systems. It may also be possible to use an emulator program to operate concurrently on a single CPU multiple incompatible operating systems. In this arrangement, although each operating system is incompatible with the other, an emulator program can host one of the two operating systems, allowing the otherwise incompatible operating systems to run concurrently on the same computer system.

When a guest computer system is emulated on a host computer system, the guest computer system is said to be a virtual machine, as the host computer system exists only as a software representation of the operation of the hardware architecture of the guest computer system. The terms emulator and virtual machine are sometimes used interchangeably to denote the ability to mimic or emulate the hardware architecture of an entire computer system. As an example, the Virtual PC software created by Connectix Corporation of San Mateo, California emulates an entire computer that includes an Intel

80X86 Pentium processor and various motherboard components and cards. The operation of these components is emulated in the virtual machine that is being run on the host machine. An emulator program executing on the operating system software and hardware architecture of the host computer, such as a computer system having a PowerPC processor, mimics the operation of the entire guest computer system. The emulator program acts as the interchange between the hardware architecture of the host machine and the instructions transmitted by the software running within the emulated environment.

A hypervisor is a control program that exists near the kernel level of a host operating system and operates to allow one or more secondary operating systems, other than the host operating system, to use the hardware of the computer system, including the processor of the computer system. A hypervisor of an operating system emulates the operating environment of the secondary operating system so that the secondary operating system believes that it is operating in its customary hardware and/or operating system environment and that it is in logical control of the computer system, when it may in fact be operating in another hardware and/or operating system environment and the host operating system may be in logical control of the computer system. Many operating systems function such that the operating system must operate as though it is in exclusive logical control of the hardware of the computer system. For multiple operating systems to function simultaneously on a single computer system, the hypervisor of each operating system must function to mask the presence of the other operating systems such that each operating system functions as though it has exclusive control over the entire computer system.

If it is desired to emulate the entire hardware architecture of the guest system, it is often useful or desirable to make maximum use of the hardware in the host computer system, including, for example, the memory management unit, the context switching hardware, the exception handling vectors, the interrupt routine pointers, and the floating point units of the host processor, and the memory of the host computer system. This hardware, however, is normally under the control of the host operating system and is shielded from access by user-level or application software. An exception is an event occurring in the computer system that cannot be resolved by the processor. An interrupt is a request sent to the processor to temporarily interrupt the current instruction stream to perform some other operation

One option to avoid this difficulty is to employ an emulation program that does not attempt to use the lower level hardware of the host computer system. This sort of approach is often used in the case of an emulation program that resides as an application program and runs on the host operating system. These application emulation programs, of course, are subject to the performance penalties of being an application program that runs on the host operating system of a host computer system. Further, these application emulation programs only gain access to the hardware features that are exposed to the application emulation program by the host operating system. As an example, an emulation program that runs as an application on the host operating system can include an exception handler. The difficulty is that the exception handler of the emulation program is separated from the processor of the host computer system by several software layers, creating a latency between the time that the host processor issues an exception and the time that the exception is identified to the exception handler of the emulation program. Another option is for the emulation program to share the hardware with the host operating system, assuming that support for the sharing of a computer system by an emulation program can be written into the hypervisor of the host operating system.

Another difficulty of using a hypervisor in an emulated computing environment is managing the memory space of the emulated computer system together with the memory space of the hypervisor itself. To improve the operation of the emulated computer system, the hypervisor will typically seek to operate the page tables of the host computer system such that the page tables of the host computer system largely mirror the page tables of the guest operating system. One difficulty of this approach is that the hypervisor code and its associated data is not mapped in to the page tables of the guest operating system. This is necessarily the case, as the emulated computer system is not aware that a hypervisor is hosting the system on the hardware of the host computer system. As such, the page table of the hypervisor does not completely match the page table of the host computer system. The page table of the hypervisor includes the mapping of the code and data of the hypervisor itself.

The mirroring by the hypervisor of the pages tables of the guest operating system raises the possibility that the address mapping of the guest operating system will conflict with the address mapping of the hypervisor, as the address mapping of the guest operating system may include the mapping of addresses over the mapped locations of the

code and data of the hypervisor itself. One solution to this problem of conflicting page mappings is to alter the page table of the guest operating system when the mapping of the logical memory of the guest operating system conflicts with the mapping prescribed by the nearly identical page table of the hypervisor. Altering the page table of the emulated computer system is undesirable, however, in that it would introduce to the emulated computer system an unexpected change to its page tables, possibly introducing other incompatibilities or faults in the guest operating system.

SUMMARY OF THE INVENTION

The present invention concerns a technique for logically decoupling a host operating system from a processor of the computer system. According to the method of the present invention, a hypervisor of an emulation program reads in a set of the processor settings of the processor. These processor settings are saved by the hypervisor to main memory. The hypervisor replaces these processor settings with a set of processor settings associated with software routines or data structures provided by the guest operating system. These software routines or data structures may include exception and interrupt handler routines and page tables. As a result of the substitution of processor settings, the host operating system is decoupled from processor control in that at least some of the processor settings of the processor are associated with software routines or data structures provided by the guest operating system, and are no longer associated with software routines or data structures provided by the host operating system. To recouple the host operating system to the processor, the hypervisor reads in the saved processor settings from memory and writes these processor settings to the appropriate registers of the processor.

During the period that the hypervisor is coupled to the processor, the page table of the hypervisor largely mirrors the page table of the guest operating system, with the primary difference being that the page table of the hypervisor includes the mapped in locations of the hypervisor code itself. To avoid conflicts between the page tables of the guest operating system and the page tables of the hypervisor, the page table of the hypervisor is reallocated if a change to the page table of the guest operating system results in mapped in memory location in the guest operating system overlapping with the mapped in location of the hypervisor code in the page table of the hypervisor.

The logical decoupling and recouping method disclosed herein is advantageous in that it allows the guest operating system to be more closely aligned with the processor for some functionality that can be provided by the guest operating system. In this manner, exception handler calls, interrupt handler calls, and memory management calls need not be passed from the processor to the host operating system before being passed to the guest operating system. Rather, these calls from the processor are passed directly to the guest operating system via the hypervisor of the emulation program.

The method disclosed herein is also advantageous in that the hypervisor or decoupling tool of the emulation program need not have access to the source code of the host operating system. Rather, the hypervisor decouples the host operating system from the processor in a manner that is transparent to the host operating system, which has no awareness that it has been decoupled or recoupled to the processor.

The method disclosed herein is also advantageous in that even if the host operating system is decoupled from the processor with respect to some functionality normally provided by the host operating system, the host operating system remains coupled to the host processor for the remainder of the functionality of the host processor. During this period, the host operating system continues to manage much of the hardware functionality of the computer system, allowing the guest operating system to route a print request through the host operating system.

An advantage of reallocating the page table of the hypervisor in the event of a conflict with the guest computer system is that the reallocation of the page table allows the page table of the hypervisor to shadow the page table of the guest operating system, while at the same time mapping in the code of the hypervisor itself. In this environment, changes to the page table of the guest operating system do not result in a conflict between the page tables of the guest operating system and the hypervisor. In addition, the reallocation of the page table of the hypervisor is transparent to the operation of the guest operating system, avoiding the possibility of creating system-level conflicts in the guest operating system.

Other technical advantages of the present invention will be readily apparent to one skilled in the art from the following figures, descriptions, and claims.

BRIEF DESCRIPTION OF THE DRAWINGS

A more complete understanding of the present invention and advantages thereof may be acquired by referring to the following description taken in conjunction with the accompanying drawings, in which like reference numbers indicate like features, and wherein:

Figure 1 is a diagram of the logical relationship of the elements of an emulated computer system running in a host computer system;

Figure 2 is a flow diagram of the method for logically coupling a hypervisor of an emulation program to the processor of a computer system;

Figure 3 is a diagram of the logical relationship of the elements of an emulated computer system in which a hypervisor is logically coupled to the processor of the computer system; and

Figure 4 is a flow diagram of a method for mapping the page tables of a guest operating system.

DETAILED DESCRIPTION OF THE INVENTION

The present invention provides an emulation environment that allows the emulation software access to the lower level hardware components of the host computer system. The invention involves an emulation program that includes a hypervisor that is transparent to the host operating system. The hypervisor of the emulation program logically disconnects or decouples the operating system from exclusive control of the host computer system for brief periods, during which time the hypervisor of the emulation program is placed in control of the hardware of the host computer system, allowing the emulation program to be logically near, for brief periods, certain hardware features of the host computer system.

The processor state of the processor is a snapshot of all of the processor settings at a single point in time. These settings include both data that the processor is manipulating as well as the program counter, pointers, and other operational flags used by the processor. Because the data being manipulated by the processor and many other processor settings may change with each processor cycle, the processor state may likewise change with each processor cycle, and may change as often as many millions of times per second.

One subset of the processor state is the user state. Taken together, the data being manipulated by the processor and the processor's program counter comprise the user state of the processor. The user state is so named because the values that comprise the user state can be manipulated by an application-level program. The remainder of the processor settings of the processor state comprise the supervisor state or privileged state of the processor. The values of the supervisor state can only be modified by software that is part of the operating system of the computer system. The settings of the supervisor state or privileged state of the processor state cannot be accessed by an application-level program. The separation of processor settings between a user state, which is accessible by application-level programs, and a supervisor state, which is not accessible by application-level programs, allows the operating system to remain functional even if the application level program crashes or causes a fatal error.

The settings of the supervisor state of the processor generally fall into one of several categories. One category of the supervisor state of the processor is the processor's mode flags, which include the settings that instruct the processor to operate according to one of several modes to support certain computational situations, exception behavior, support for legacy programs, or the selective use of newer features of the processor. Another category of settings of the processor state are those settings that control external hardware access. Processors often support several operational settings for communication between the processor and hardware external to the processor, such as caches, memory, and other input and output devices. Because these operational settings, including the communication speed and bandwidth, for example, can only be modified by privileged or supervisor level code, they are considered to be part of the supervisor state of the processor.

Another category of settings of the privileged state of the processor are the exception and interrupt routine pointers. When the processor encounters an exception or is asked for an interrupt, the processor stops execution at the current program counter address and begins execution at a predetermined exception routine address. The location of these exception routines are identified through the use of exception routine pointers. For some processors, the exception routines are located at hard-coded memory addresses. On most processors, however, their locations are programmable and the value of the locations are considered to be part of the supervisor state of the processor.

A set of data structure pointers make up another category of the privileged state of the processor. Because some data structures of the processor state are too large to be stored in the processor, these data structures are stored in main memory and a pointer to the data structure is stored at the processor. An example of this sort of large data structure is the processor's page tables. These tables define for the processor the logical-to-physical translation of memory addresses that are necessary for the conversion of virtual memory accesses to physical memory locations. Both the page table and the pointer to the page table, which is stored at the processor, are considered to be part of the supervisor state of the processor. Another example of a data structure of the processor's supervisor state that is stored in main memory is an exception routine pointer table. Rather than storing the pointer for each exception routine at the processor, a single pointer stored at the processor can point to a table of exception routine pointers. The exception pointer or vector table will include a listing of address pointers to the exception handlers accessible by the processor. Both the exception routine pointer that is stored at the processor and the exception routine pointer table are considered part of the supervisor state.

The host operating system typically establishes for the processor the vectors or pointers to the exception and interrupt handler routines. If, for example, the processor is instructed to process a set of commands that will cause the processor to divide by zero, the processor will call the exception handler routine that handles the divide-by-zero exception. In doing so, the processor may call an exception handler routine that is pointed to by a vector stored at the processor. Alternatively, the processor may scan the exception vector table for the correct exception handler for the exception condition and then call the exception handler using the exception vector or pointer associated with the applicable exception handler.

When the host operating system is in logical control of the host processor, the host operating system defines for the host processor the exception vectors and all other vectors that will be needed by and are supported by the host processor. Should the processor encounter an exception condition or need to access another routine that is referenced by a vector, such as the page table routines and the page table vector, the processor executes the host operating system's exception handler.

In the case of an emulated computer system, an emulation program provides an emulated operating environment in the host computer system. Shown in Figure 1 is a diagram of the logical layers of the hardware and software architecture for an emulated operating environment in a computer system 10. An emulation program 14 runs on a host operating system that executes on the host computer system hardware or processor 11. Emulation program 14 emulates a guest computer system 16, including a guest operating system 18. Guest application programs are able to execute on guest operating system 18. In the emulated operating environment of Figure 1, because of the operation of emulation program 14, guest application 20 can run on the computer system 10 even though guest application 20 is designed to run on an operating system that is generally incompatible with host operating system 12 and host computer system hardware 11. In the architecture of Figure 1, guest operating system 18 is separated from the host computer system hardware 11 across several logical and software layers, including the host operating system 12. This logical separation introduces latency and performance difficulties in those instances in which the guest operating system attempts to communicate directly with the host computer system hardware 11.

In the example of Figure 1, host operating system 12 is in exclusive operating control of the host computer system hardware 11, including the processor of the host computer system. The host operating system will have established for the processor of the host computer system a number of the settings of the processor state. For example, the host operating system 12 may have established a set of exception handler vectors, whether located in a vector table or in the registers of the host processor, a set of interrupt handlers, and a page table vector. Thus, when the processor makes exception handler calls, interrupt handler calls, or memory management calls, the processor settings for these calls is provided on an exclusive basis by the host operating system. Because the host operating system established the settings of the supervisor or privileged state of the processor, the host operating system is said to be in logical control of the processor of the host computer system.

According to the emulation technique of the present invention, the emulation program itself includes a hypervisor that is able to logically disconnect or decouple the host operating system from exclusive control of the processor and other host computer system hardware 11. The hypervisor of emulation program 14 accomplishes the

logical disconnection step by reading in and saving to a memory location the vectors and other address pointers that are used by the host processor. A flow diagram of the steps taken by the hypervisor to logically disconnect the host operating system from the exclusive control of the host computer system and to logically connect or couple the hypervisor of the emulation program to the host computer system is shown in Figure 2. When the hypervisor of the emulation program wants to assert control over at least some of the processor settings of the processor, the hypervisor at step 24 reads in some or all of the settings of the supervisor or privileged state of the processor. According to one embodiment of the invention, the processor reads in the address pointers from the applicable registers of the processor, including the exception vectors or the pointer to the exception vector table, the interrupt pointers or the pointer to the interrupt routine table, and the pointer to the page table. At step 26, the hypervisor saves these processor settings, together with an identification of their associated storage location on the host processor, to memory. The hypervisor next substitutes at step 28 the user-level processor state of the guest operating system and the privileged-level state of the hypervisor of the emulation program. As a result, the processor settings of the host processor are replaced with a combination of the user-level processor state of the guest operating system and the privileged-level state associated with the hypervisor of the emulation program. In doing so, the hypervisor stores in the appropriate registers of the processor the vectors and other address pointers to the exception handlers, interrupt routines, and page maps of the hypervisor of the emulation program and the guest operating system.

Because the processor settings that are substituted by the hypervisor of the emulation program are supervisor or privileged level settings, the hypervisor of the emulation program has logical control of the processor for all of the substituted processor settings. In this manner, the emulated computer system is logically in control of the hardware functionality of the host computer system for the functionality governed by the substituted processor settings. A diagram of the logical relationship of the emulation program 14 following substitution of the processor settings is shown in Figure 3. Hypervisor 22 in Figure 3 is shown as being a component of emulation program 14, which is shown as having a logical communication link to the processor and host hardware 11.

As an example of the emulation program's logical control of host hardware 11, if an exception were to occur during this period, the processor would call an exception

handler provided by the hypervisor of the emulation program. Similarly, if the processor is called on to translate a virtual memory address to a physical memory address, the processor will access the page table associated with the guest operating system. Thus, many settings of the processor states have been replaced so that the processor is calling upon or accessing software or data structures provided directly by the guest operating system. This allows the guest operating system to logically align itself more closely with the processor of the computer system, and eliminates the latency caused by separating the guest operating system from the processor of the computer system across the host operating system. In the case of an exception, the exception handler of the hypervisor may handle the exception itself or the exception handler of the hypervisor may invoke the exception handler of the guest operating system, allowing the guest operating system to directly handle the exception without the necessity of having the exception call pass through the host operating system software layer.

As shown in Figure 3, host operating system 12 is not permanently disconnected form the hardware of the host computer system. The logical control asserted by the hypervisor over a set of processor settings does not result in hypervisor 22 assuming responsibility over all the hardware interaction in the computer system. During those periods when the host operating system is handling a function request relating to the hardware of the computer system, the host operating system must be in logical control of the computer system. To accomplish this, the host operating system must be temporarily logically recoupled to the hardware of the computer system. Once the host operating system has completed the functions requested by the emulation program, the emulation program can reassume its logical control over the exception and interrupt handlers associated with the processor setting of the emulation program.

With regard to the flow diagram of Figure 2, at step 30, hypervisor 22 returns the logical control over the processor settings to the host operating system. Hypervisor 22 retrieves from memory the vectors and other pointers that were saved to memory. These addresses are written back to the appropriate register locations, returning logical control over these processor settings to the host operating system. During the period that the host operating system is logically disconnected from the processor, the logical disconnection of host operating system 12 is not apparent to host operating system 12. Host operating system 12 is not aware that a set of processor settings has been

substituted, resulting in the transfer of logical control for some hardware functionality to the emulation program. When logical control is returned to host operating system 12, host operating system 12 is similarly not aware that logical control of a set of hardware calls has been returned to it.

The substitution of some of the processor settings of the processor supervisor states results in a performance gain by the emulation program as compared with the option of the handling by the emulation program of exceptions, interrupts, and memory management functions as filtered through the host operating system. In those cases in which the emulation program operates as an application program that resides on the host operating system, logically disconnecting the host operating system from exclusive control over the functionality of the processor is a performance advantage for the emulated computer system. Once the host operating system has been logically decoupled from at least some of the functionality of the processor, the emulation program, by substituting its processor settings for the processor settings of the host operating system, can operate much more closely with the host processor. Once the processor settings have been substituted, the emulation program can direct the processor to the emulation program's own set of exception vectors, interrupt routines, and page tables, thereby avoiding the latency problems caused when the host operating system separates the processor from the emulation program.

The interrupt and exception handling functions of the hypervisor of the present invention also permit the handling of exceptions and interrupts as expected by the guest operating system. An operating system may choose to handle certain exceptions at the operating system level without passing these exceptions to the application that initiated the exception. One example is the divide-by-zero exception, which is often handled by the operating system without being passed to the application program that initiated the exception. The hypervisor of the emulation program can continue to pass exceptions to the guest operating system without the necessity of initiating an exception handler in the hypervisor. Thus, as expected, when a guest application program causes the initiation of a divide-by-zero exception at a time when the emulation program is in logical control of the processor with respect to the substituted processor settings, the hypervisor of the emulation program is able to pass the exception to the guest operating system.

Shown in Figure 4 is a flow diagram of a method for mapping the page tables of a guest operating system. At step 40, the hypervisor identifies a write to the page table of the guest operating system. At step 42, the hypervisor determines if the write to the page table of the guest operating system resulted in mapping conflict between the page tables of the guest operating system and the page table of the hypervisor. The page table of the hypervisor closely mirrors the page table of the guest operating system, with the principal difference being that the page table of the hypervisor maps in to its page table the code and data of the hypervisor. The hypervisor determines that there is a conflict, for example, if mapped pages in the page table of the guest operating system overlap with the mapped in locations of the hypervisor in the page tables of the hypervisor.

If the hypervisor determines that there will be no conflict caused by the change to the page tables of the guest operating system, then no change is necessary to the page tables of the hypervisor (step 44). If the hypervisor determines that a change to the page table of the guest operating system causes a conflict with the page table of the host computer system, the hypervisor reallocates its page table at step 46 so that the code and data of the hypervisor itself is moved or mapped in to another location in the page table of the hypervisor. This process of identifying a change to the page table of the guest operating system and, if necessary, reallocating the page table of the hypervisor in response, is transparent to the operation of the guest operating system. Once the page table of the hypervisor has been reallocated, if necessary, the page table of the guest operating system no longer conflicts with the page table of hypervisor.

The present invention is not limited in its application to the emulation of a particular computer system architecture, particularly the Intel 80X86 architecture. Rather, the emulation technique disclosed herein is applicable any time it is desirable that a host operating system be transparently disconnected from the processor with respect to at least some of the processor settings of the processor.

Although the present invention has been described in detail, it should be understood that various changes, substitutions, and alterations can be made thereto without departing from the spirit and scope of the invention as defined by the appended claims.

WHAT IS CLAIMED IS:

1. A method for emulating in a host computer system a guest computer system, the guest computer system including a guest operating system, comprising the steps of:

establishing in the host computer system a hypervisor for hosting a guest operating system;

establishing in the hypervisor a page table;

monitoring the page table of the guest operating system; and

reallocating the page table of the hypervisor if a change to the page table of the guest operating system causes a conflict between the page table of the hypervisor and the page table of the guest operating system.

2. The method for emulating in a host computer system a guest computer system, the guest computer system including a guest operating system of claim 1, wherein the page table of the hypervisor generally mirrors the page table of the guest operating system, except for the inclusion in the page table of the hypervisor of the mapping of the code of the hypervisor.

3. The method for emulating in a host computer system a guest computer system, the guest computer system including a guest operating system of claim 1, wherein the step of monitoring the page table of the guest operating system comprises the step of monitoring for writes to the page table of the guest operating system.

4. The method for emulating in a host computer system a guest computer system, the guest computer system including a guest operating system of claim 1, wherein the step of reallocating the page table of the hypervisor comprises the step of reallocating the mapping of the code of the hypervisor such that the code of the hypervisor is not mapped in to a memory location that overlaps with a mapped in memory location of the page table of the guest operating system.

5. The method for emulating in a host computer system a guest computer system, the guest computer system including a guest operating system of claim 1,

wherein the page table of the hypervisor generally mirrors the page table of the guest operating system, except for the inclusion in the page table of the hypervisor of the mapping of the code of the hypervisor;

wherein the step of monitoring the page table of the guest operating system comprises the step of monitoring for writes to the page table of the guest operating system; and

wherein the step of reallocating the page table of the hypervisor comprises the step of reallocating the mapping of the code of the hypervisor such that the code of the hypervisor is not mapped in to a memory location that overlaps with a mapped in memory location of the page table of the guest operating system.

6. A method for reconciling, in an emulated computing environment, the page table of a guest operating system with the page table of a hypervisor, wherein the hypervisor is hosting the guest operating system on the host computer system, comprising the steps:

establishing the page table of the hypervisor such that the page table of the hypervisor largely mirrors the page table of the guest operating system, with the addition that the page table of the hypervisor includes an allocation of memory space for the code of the hypervisor;

monitoring the page table of the guest operating system for changes to the page table of the guest operating system;

if a change to the page table of the guest operating system causes a conflict between the page table of the hypervisor and the page table of the guest operating system, reallocating the page table of the guest operating system so that there is not a conflict between the page table of the hypervisor and the page table of the guest operating system.

7. The method for reconciling, in an emulated computing environment, the page table of a guest operating system with the page table of a hypervisor of claim 6, wherein the step of reallocating the page table of the guest operating system comprises the step of reallocating the page mappings of the code of the hypervisor such that the page mappings of the code of the hypervisor do not overlap with any mapped in pages in the guest operating system.

8. The method for reconciling, in an emulated computing environment, the page table of a guest operating system with the page table of a hypervisor of claim 7, wherein the step of monitoring the page table of the guest operating system comprises the step of determining when there has been a write to the page table of the guest operating system.

9. A method for emulating in a host computer system a guest computer system, the host computer system including a processor and a host operating system, comprising the steps of:

running an emulator program on the host system that emulates the operation of a guest operating system;

reading in from the processor of the computer system a processor setting associated with the host operating system;

writing the processor setting to memory;

writing to the processor a guest processor setting associated with the emulator program;

wherein the host operating system is logically decoupled from the processor for the processor function related to the guest processor setting associated with the emulator program.

10. The method for emulating in a host computer system a guest computer system of claim 9,

wherein the processor setting associated with the emulator program is an exception vector; and

further comprising the step of calling the exception handler pointed to by the exception vector.

11. The method for emulating in a host computer system a guest computer system of claim 9, further comprising the steps of,

reading in from memory the processor setting associated with the host operating system;

writing to the processor the processor setting associated with the host operating system, wherein the host operating system is logically coupled to the processor for the function related to the processor setting associated with the host operating system.

12. The method for emulating in a host computer system a guest computer system of claim 9, further comprising the steps of:

temporarily logically recoupling the host operating system to the processor for the purpose of performing a function requested by the emulator program; and

logically recoupling the emulator program to the processor for the function related to the processor setting associated with the emulator program following the completion by the host operating system of the function requested by the emulator program.

13. The method for emulating in a host computer system a guest computer system of claim 9, wherein the emulator program operates as an application program on the host operating system.

14. The method for emulating in a host computer system a guest computer system of claim 9,

wherein the processor setting associated with the emulator program is an interrupt routine pointer; and

further comprising the step of calling the interrupt routine pointed to by the interrupt routine pointer.

15. The method for emulating in a host computer system a guest computer system of claim 9,

wherein the processor setting associated with the emulator program is a pointer to an exception vector table; and

further comprising the step of accessing the exception vector table using the processor setting associated with the emulator program.

16. The method for emulating in a host computer system a guest computer system of claim 9,

wherein the processor setting associated with the emulator program is a pointer to a page table associated with the guest computer system; and

further comprising the step of accessing the page table of the guest computer system using the processor setting associated with the emulator program.

17. A method for logically decoupling a host operating system of a computer system from the processor of the computer system, comprising the steps of:

running as an application program on the host computer system an emulator program that emulates the operation of a guest operating system, the guest operating system having a set of functionality associated with the supervisor state of the processor of the computer system;

replacing in the processor as part of a first replacement step a processor setting associated with the functionality of the guest operating system and the supervisor state of the processor such that the host operating system is logically decoupled from the processor with respect to the processor operations associated with the replaced processor setting; and

replacing in the processor as part of a second replacement step a processor setting associated with the functionality of the host operating system such that the host operating system is logically recoupled to the host operating system with respect to the processor operations associated with the replaced processor setting.

18. The method for logically decoupling a host operating system of a computer system from the processor of the computer system of claim 17, wherein the processor setting associated with the functionality of the guest operating system is a pointer to an exception handler provided by the guest operating system.

19. The method for logically decoupling a host operating system of a computer system from the processor of the computer system of claim 17, wherein the processor setting associated with the functionality of the guest operating system is a pointer to a page table of the guest operating system.

20. The method for logically decoupling a host operating system of a computer system from the processor of the computer system of claim 17, further comprising the steps of:

temporarily logically recoupling the host operating system to the processor for the purpose of performing a function requested by the emulator program; and

logically recoupling the emulator program to the processor for the processor settings associated with the guest operating system following the completion by the host operating system of the function requested by the emulator program.

21 The method for logically decoupling a host operating system of a computer system from the processor of the computer system of claim 17, wherein the first replacement step comprises the steps of,

reading in from an identifiable register of the processor a processor setting associated with the host operating system; and

writing the processor setting read in from memory to memory.

22. The method for logically decoupling a host operating system of a computer system from the processor of the computer system of claim 21, wherein the second replacement step comprises the steps of,

reading in from memory the processor setting associated with the host operating system; and

writing the processor setting to the identifiable register of the processor.

23. The method for logically decoupling a host operating system of a computer system from the processor of the computer system of claim 21, wherein the memory is RAM memory.

24. The method for logically decoupling a host operating system of a computer system from the processor of the computer system of claim 17, wherein the processor setting associated with the functionality of the guest operating system is a pointer to an exception handler provided by the guest operating system.

25. A method for rerouting calls by a processor of a computer system to a guest operating system being emulated on a host operating system of the computer system, comprising the steps of:

reading in a first set of processor settings from the processor, the processor settings being associated with functionality provided by the host operating system;

storing the first processor settings in memory;

writing in a second set of replacement processor settings to the processor, the replacement set of processor settings associated with the functionality provided by the guest operating system, wherein with respect to the functionality associated with the replacement set of processor settings, the host operating system is logically decoupled from the processor and the guest operating system is logically coupled to the processor.

26. The method for rerouting calls by a processor of a computer system to a guest operating system being emulated on a host operating system of the computer system of claim 25, further comprising the steps of,

reading in the first set of processor settings stored in memory; and

writing the first set of processor settings to the processor, wherein with respect to functionality associated with the first set of processor settings, the host operating system is logically coupled to the processor.

27. A method for logically decoupling a host operating system from a processor of a computer system, comprising the steps of:

saving the contents of a set of registers on the processor to memory, the set of registers storing at least a part of the processor settings of the processor; and

writing to the set of registers a replacement set of processor settings associated with functionality provided by a guest operating system that resides as an application program on the host operating system.

28. The method for logically decoupling a host operating system from a processor of a computer system of claim 20, wherein the replacement set of registers includes pointers to exception handler routines provided by the host operating system.

29. The method for logically decoupling a host operating system from a processor of a computer system of claim 27, wherein the replacement set of registers includes pointers to interrupt handler routines provided by the host operating system.

30. The method for logically decoupling a host operating system from a processor of a computer system of claim 27, wherein the replacement set of registers includes a page table associated with the host operating system.

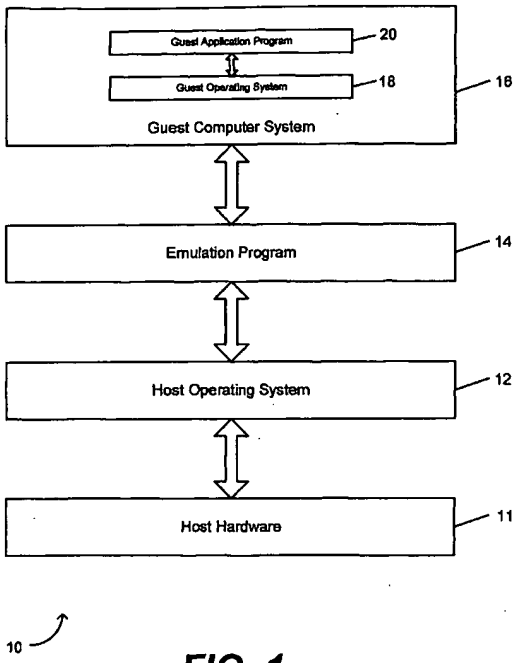

Guest Application Program
20
Guest Operating System
18
16
Guest Computer System
Emulation Program
14
Host Operating System
12
Host Hardware
11
10

FIG. 1

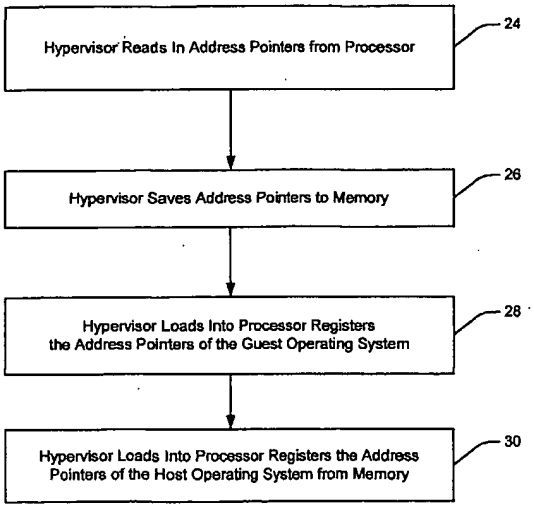

Hypervisor Reads In Address Pointers from Processor
24
Hypervisor Saves Address Pointers to Memory
26
Hypervisor Loads Into Processor Registers the Address Pointers of the Guest Operating System
28
Hypervisor Loads Into Processor Registers the Address Pointers of the Host Operating System from Memory
30

FIG. 2

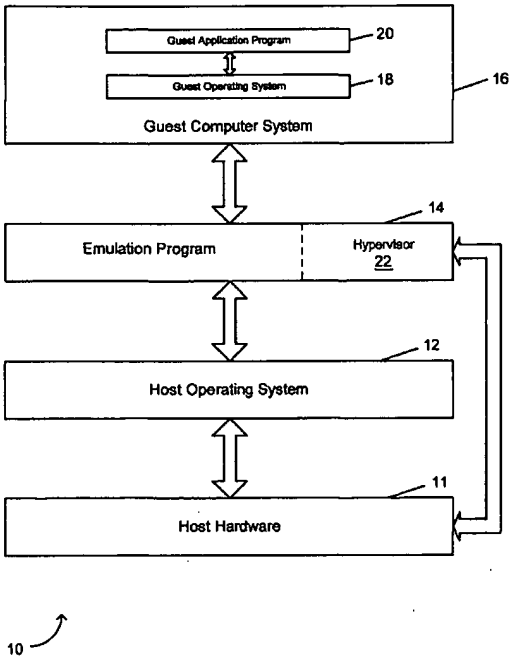

Guest Application Program
20
Guest Operating System
18
16
Guest Computer System
14
Emulation Program
Hypervisor
22
12
Host Operating System
11
Host Hardware
10

FIG. 3

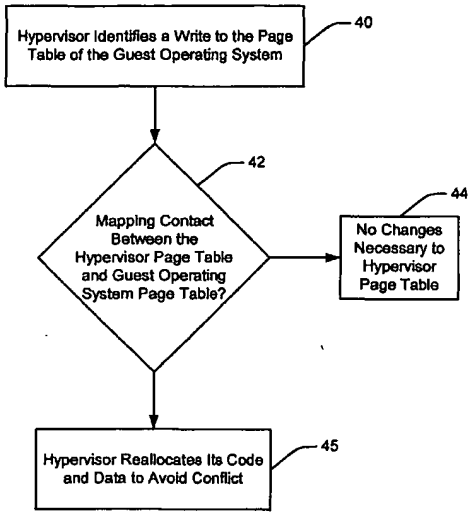

Hypervisor Identifies a Write to the Page Table of the Guest Operating System
40
Mapping Contact Between the Hypervisor Page Table and Guest Operating System Page Table?
42
No Changes Necessary to Hypervisor Page Table
44
Hypervisor Reallocates Its Code and Data to Avoid Conflict
45

FIG. 4

【国際公開パンフレット（コレクトバージョン）】

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization International Bureau

(43) International Publication Date
27 June 2002 (27.06.2002)

PCT

(10) International Publication Number
WO 2002/050664 A3

(51) International Patent Classification⁷: G06F 9/455

(21) International Application Number:
PCT/US2001/049217

(22) International Filing Date:
19 December 2001 (19.12.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/747,492 21 December 2000 (21.12.2000) US
09/906,392 16 July 2001 (16.07.2001) US

(71) Applicant: CONNECTIX CORPORATION [US/US]; 2955 Campus Drive, Suite 100, San Mateo, CA 94403 (US).

(72) Inventor: TRAUT, Eric, P.; 3 Iris Lane, San Carlos, CA 94070 (US).

(74) Agent: FULGHUM, Roger, J.; Baker Botts L.L.P., One Shell Plaza, 910 Louisiana, Houston, TX 77002 (US).

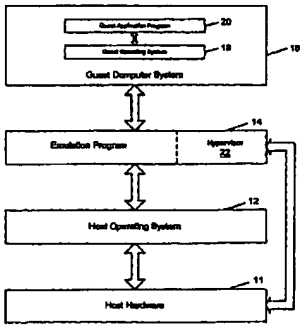
(81) Designated States (national): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States (regional): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— with international search report

[Continued on next page]

(54) Title: LOGICAL SUBSTITUTION OF PROCESSOR CONTROL IN AN EMULATED COMPUTING ENVIRONMENT



(57) Abstract: In an emulated computing environment, a method is provided for logically decoupling the host operating system from the processor of the computer system with respect to certain processor settings of the processor. A hypervisor of the emulation program replaces some of the processor settings of the processor with processor settings associated with software routines or data structures provided by the guest operating system. During this period, when the processor calls a software routine or accesses a data structure associated with the replaced processor setting, the processor will call or access a software routine or access a data structure associated with the guest operating system, bypassing the host operating system and communicating directly with the guest operating system. When the host operating system is to be recoupled to the processor, the processor settings that have been saved to memory are rewritten to the appropriate registers of the processor.

WO 2002/050664 A3

WO 2002/050664 A3

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(88) Date of publication of the international search report:
26 February 2004

【国際調査報告】

INTERNATIONAL SEARCH REPORT

International Application No
PCT/US 01/49217

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F9/455

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	DE 42 17 444 A (HITACHI LTD ;HITACHI COMPUTER ENG (JP)) 3 December 1992 (1992-12-03) column 2, line 66 -column 3, line 59 column 6, line 31 -column 7, line 15 column 5, line 13 - line 27	1-8
X	FR 2 587 519 A (NIPPON ELECTRIC CO) 20 March 1987 (1987-03-20) claim 1	9-30

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
12 December 2003	19/12/2003

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Palencia Gutiérrez,C

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/US 01/49217

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
DE 4217444	A	03-12-1992	JP	4348434 A	03-12-1992
			DE	4217444 A1	03-12-1992
			GB	2256513 A ,B	09-12-1992
FR 2587519	A	20-03-1987	JP	2016787 C	19-02-1996
			JP	7031609 B	10-04-1995
			JP	62066336 A	25-03-1987
			FR	2587519 A1	20-03-1987

Form PCT/ISA/210 (patent family annex) (July 1992)

フロントページの続き

(81)指定国 AP(GH,GM,KE,LS,MW,MZ,SD,SL,SZ,TZ,UG,ZM,ZW),EA(AM,AZ,BY,KG,KZ,MD,RU,TJ,TM),EP(AT,BE,CH,CY,DE,DK,ES,FI,FR,GB,GR,IE,IT,LU,MC,NL,PT,SE,TR),OA(BF,BJ,CF,CG,CI,CM,GA,GN,GQ,GW,ML,MR,NE,SN,TD,TG),AE,AG,AL,AM,AT,AU,AZ,BA,BB,BG,BR,BY,BZ,CA,CH,CN,CO,CR,CU,CZ,DE,DK,DM,DZ,EC,EE,ES,FI,GB,GD,GE,GH,GM,HR,HU,ID,IL,IN,IS,JP,KE,KG,KP,KR,KZ,LC,LK,LR,LS,LT,LU,LV,MA,MD,MG,MK,MN,MW,MX,MZ,NO,NZ,PL,PT,RO,RU,SD,SE,SG,SI,SK,SL,TJ,TM,TR,TT,TZ,UA,UG,UZ,VN,YU,ZA,ZW

【要約の続き】
ハイパーバイザがプロセッサに結合されている時間中は、ハイパーバイザのページテーブルは、ゲストオペレーティングシステムのページテーブルを大部分ミラーリングするように確立される。ゲストオペレーティングシステムのページテーブルへの変更により、ゲストオペレーティングシステムのページテーブルとハイパーバイザのページテーブルの競合が起きた場合は、ハイパーバイザのページテーブルは、ゲストオペレーティングシステムに対してトランスペアレントな方式で再割り当てされる。